SÓLO PROGRAMADORES

Revista especializada para usuarios de PC

AÑO 2. Nº 13
1250 PTAS.

ARGENTINA 9'50 $
CHILE 3000 $

Concurrencia en MS-DOS

Visual Basic vs Delphi

Sonido Estéreo en 16 bits

TCP/IP para LINUX

Y además:

- Listado de BBS's nacionales
- Las shells de Unix
- Programación bajo Windows
- Construcción de demos
- Programación C++

TORNEO DE PROGRAMACIÓN DE DEMOS

250.000 Ptas. en efectivo para el ganador

DE REGALO
CD-ROM con
Velocis 1.3.0

9 778411 344792 00013

TOWER COMMUNICATIONS, S.R.L.

MINISTERIO DE DEFENSA
DIRECCION GENERAL DE LA GUARDIA CIVIL
MINISTERIO DE CULTURA
MINISTERIO DE ECONOMIA Y HACIENDA
MINISTERIO DE INDUSTRIA, COMERCIO Y TURISMO
GENERALITAT DE CATALUNYA. DEP. SANITAT
SERVEI CATALA DE LA SALUT
BOLETIN OFICIAL DEL ESTADO
M.O.P.U.

BANCO DE SANTANDER
INSTITUTO NCNAL. DE LA SEG. SOCIAL
MINISTERIO PARA LAS ADMINISTRACIONES PUBLICAS
FONDO DE GARANTIA DE DEPOSITOS
BANCO SANTANDER PUERTO RICO
BANCO ATLANTICO
PRICE WATERHOUSE
TOSHIBA
I.N.E.M.

SANTANDER NATIONAL BANK
CORPORACION FINANCIERA HISPAMER
ANALISTAS FINANCIEROS INTERNACIONALES
SEGUROS OCASO
EUROSEGUROS BBV
SEGUROS ATHENA
FYCSA (ALCATEL)
SECRETARIA DE ESTADO PARA LAS COMUNIDADES EUROPEAS

CONSTRUCCIONES AERONAUTICAS (CASA)
PATENTES TALGO
SINTEL
AVIACO
TRANSMEDITERRANEA
PUERTOS DEL ESTADO
GEC ALSTHOM
L´OREAL
NATIONALE NEDERLANDEN

FASA RENAULT
RED DE CONCESIONARIOS RENAULT
REPSOL EXPLORACION
REFINERIA DE GIBRALTAR
PETROGAL
ONDA CERO RADIO
TELEMADRID
ANTENA 3 TELEVISION
A.B.B. (ASEA BROWN BOVERI)
IVECO - PEGASO

TELECINCO (GESTEVISION-TELECINCO)
UNIVERSIDAD AUTONOMA DE MADRID
UNIVERSIDAD CARLOS III DE MADRID
INSTITUTO DE EMPRESA
TELEFONICA SISTEMAS*
S.P. EDITORES*
OLIVETTI*
SOFTWARE AG
JUNTA DE EXTEMADURA

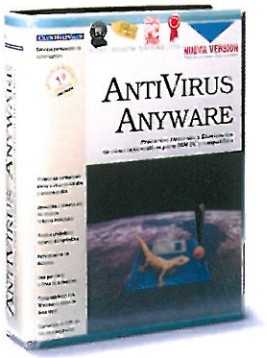
IBM*
SIEMENS NIXDORF*
DIGITAL*
BULL ESPAÑA*
FUJITSU*
INVESTRONICA*
SOFTWARE DE DIAGNOSTICO*
SEUR
CAJA LABORAL POPULAR

TELEFONICA
INFORMATICA EL CORTE INGLES*
ACTION*
GTI*
ALCATEL SISTEMAS DE INFORMACION*
INDAS
DELOITTE & TOUCHE

EL CORTE INGLES*
SOFTWARE DE ESPAÑA*
INFORMIX
ABBOTT CIENTIFICA
ALBILUX
ELIDA GIBBS
OXFORD UNIVERSITY PRESS
A.E.N.A. (AEROPUERTOS ESPAÑOLES Y NAVEGACION AEREA)

9 de cada 10 Ordenadores prefieren ANTIVIRUS ANYWARE.

El resultado está a la vista. O, ¿cree que tantas Empresas pueden estar equivocadas?. Tienen razones para no estarlo.



1. MAYOR PROTECCION.

Incorpora un Sistema de Protección con una ocupación mínima en RAM. Detecta el Virus antes de que pueda introducirse e impide el uso del fichero contaminado, avisándole a través de una ventana de alarma.

2. DETECCION INSTANTANEA.

De forma rápida y precisa, analiza cualquier unidad, directorio o fichero. Comprueba si su disco duro o disquetes contienen algún Virus. Chequea ficheros (incluso comprimidos), sector de arranque, tabla de particiones y unidades de red.

3. ELIMINACION DEFINITIVA.

Elimina los Virus allí donde se encuentren, sin dañar los ficheros.

4. ACTUALIZACION PERMANENTE.

Usted puede recibir las nuevas versiones cómodamente en su domicilio o capturarlas vía modem.

ANYWARE pone a su disposición el CLUB DE USUARIOS HELP VIRUS con una HOT LINE exclusiva para consultas, noticias, BBS, etc.

PREMIO PC MAGAZINE MEJOR UTILIDAD 1991 — RECOMENDACION PCWORLD MEJOR PRODUCTO ANTIVIRUS — RECOMENDACION PCMAGAZINE MEJOR PRODUCTO ANTIVIRUS — CATACOM 93 MEJOR ANTIVIRUS — PREMIO PC MAGAZINE MEJOR UTILIDAD 1993

ANYWARE
SEGURIDAD INFORMATICA

Orense, 36 3º. 28020 MADRID. España.
Tel.: (91) 556 92 15. Fax.: (91) 556 14 04.

(*)EMPRESAS DISTRIBUIDORAS DE ANTIVIRUS ANYWARE.
EMPRESAS E INSTITUCIONES CON LICENCIA CORPORATIVA DE ANTIVIRUS ANYWARE

SÓLO PROGRAMADORES
Revista práctica para usuarios de PC

CARTA DEL DIRECTOR

¿QUÉ FUE DEL CREADOR DE LA MANZANA?

Paso ya mucho tiempo desde que Steven Jobs, cofundador de Apple, dejará la empresa que, durante mucho tiempo, fue la única alternativa a IBM (en EEUU, por supuesto) en el mercado de los ordenadores personales. Tal vez muchos de vosotros conozcáis su historia, para los que no, la contaré a continuación de forma resumida. Wozniak, Stenven Jobs y Makkula crearón antes de 1980 el mítico Apple I en el garaje de la casa de Wozniak. Jobs vendió su furgoneta para conseguir fondos para la fabricación de las primeras unidades y Makkula, puso la mentalidad comercia y el capital inicial para el comienzo de la producción en serie. El crecimiento inicial de Apple Computer fue tan expectacular que en pocos años estaba en los primeros puestos entre las empresas citadas en la prestigiosa revista financiera FORTUNE.

Apple lideraba el mercado del ordenador doméstico muy por encima de sus competidores directos (commodore, Atari, Timex -Sinclair en EEUU, ¿recordáis al *tío* Clive?-) y parecía que Jobs y Apple no tendrían techo en este planeta. Incluso en un momento de *delirio* o *genialidad* (según para quién) llegaron a fichar a John Sculley (el hombre que había sido el artífice del milagro de Pepsi frente a Coca Cola) como Director General de Apple Computer, por un módico sueldo de un millón de dolares anuales, hombre que a la larga llegaría a expulsar a Jobs de su propia empresa. Lo realmente importante es que cuando Jobs concibió el Macintosh, las capacidades de su sistema operativo estaban muy por encima de cualquier MS DOS para PC.

Los usuarios de Macintosh llevan varios años trabajando en *Windows* sin saber que un día alguien presentaría Windows 95 como la panacea de los Sistemas Operativos. Y aquí es donde viene la gran pregunta ¿qué fue del creador de la manzana? ¿por qué Jobs no vislumbró un futuro en el que la informática estaría a merced del software y no del hardware? Apple Computer tuvo durante varios años la llave de un sistema operativo que superaba a los equivalentes MS DOS de la época. No lo licenció para PC porque entendió que era vender su más preciada joya a los ordenadores que hacían la competencia a la serie Mac y Apple II, dejando el hueco de mercado que todos conocemos. Este hueco fue aprovechado por Bill Gates y Microsoft.

Hoy en día los Power Mac son casi compatibles PC y lo más irónico es que incluso ejecutan Windows, y dentro de poco, Windows 95. Seguro que Steven Jobs, desde Next Computers (su actual empresa) aún sueña con el momento en el que decidió no licenciar su sistema operativo para PC. De haberlo hecho así, hoy windows no existiría y Gates no sería el hombre más rico de EEUU. El mundo apple y el de los PCs llevarían 9 años interconectados a través de un sistema operativo cuyo genial creador, no fue capaz de ver ni supo explotar la magnitud del concepto que había inventado.

Moraleja: las ideas, las oportunidades y, sobre todo, la posibilidad de cambiar la historia, se da pocas veces en la vida. Cada cual lo aplique a donde crea oportuno...

MARIO DE LUIS

SÓLO PROGRAMADORES
Revista práctica para usuarios de PC

SEPTIEMBRE 1995. Número 13
Es una publicación de

TOWER COMMUNICATIONS, S.R.L.

Editor
Antonio M. Ferrer Abelló
Director
Mario de Luis García
Redactor Jefe
Carlos Doral Pérez
Coordinador técnico
Eduardo Toribio Pazos

••••••

Directora Comercial
Carmina Ferrer
Director de Producción
Carlos Peropadre
Publicidad
Magdalena Pedreño Llorente

Colaboradores
Emilio Postigo, Fernando J. Echevarrieta, Juan M. Martín, José María Peco, José Carlos Remiro, Pedro Antón, Luis Crespo, Enrique de Alarcón, Jorge R. Regidor, Agustín Guillén, Enrique Coiras, David Aparicio, Fernando de la Villa, Ignacio Cea, Enrique Velasco.

Edición técnica
Emilio Castellano
Maquetación
Fernando García Santamaría
Tratamiento de imagen
Josefa Fernández Martínez
Servicios Informáticos
Digital Dreams Multimedia, S.L.
Ilustraciones
Miguel Alcón
Secretaría de Redacción
Rosa Arroyo
Suscripciones
Erika de la Riva
Redacción, Publicidad y Administración
C/ Marqués de Portugalete, 10
28027 MADRID
Telf.: 741 26 62 / Fax: 320 60 72
Filmación
Duvial
Impresión
G.D.B.
Distribución
MIDESA

La revista SÓLO PROGRAMADORES no tiene por qué estar de acuerdo con las opiniones escritas por sus colaboradores en los artículos firmados.
El editor prohibe expresamente la reproducción total o parcial de los contenidos de la revista sin su autorización escrita.

Depósito legal: M-26827-1994
ISSN: 1134-4792

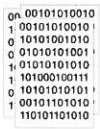
SUMARIO

6 **CONCURSO**
Sólo Programadores convoca el primer concurso de demos e infografía. En este número se encuentra el cupón para poder participar.

8 **NOTICIAS**
Este es el espacio dedicado a informar al lector de las últimas novedades del mundo de la informática y el comentario de los libros de interés.

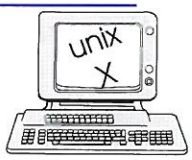
12 **HERRAMIENTAS VISUALES**
Un análisis comparativo de las dos herameintas que mayor desarrollo y difusión han tenido en los últimos meses, Visual Basic y Delphi.

17 **CURSO DE ENSAMBLADOR**
En esta última entrega se explican las macros, una herramienta de programación que hace de la elaboración de programas una tarea más sencilla.



22 **CURSO DE PROGRAMACIÓN**
En este número se explica de manera pormenorizada una de las estructuras de datos más utilizadas en programación, los arrays.

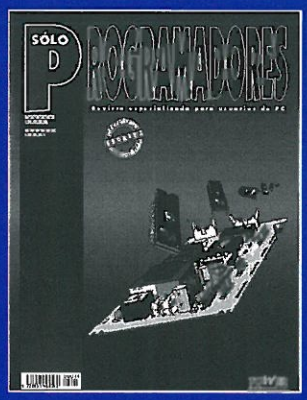
26 **CURSO DE UNIX**
Con la presente entrega de este curso se pretende que el lector aprenda a configurar y personalizar su entorno, tanto si es usuario como root del sistema.



32 **TRATAMIENTO DIGITAL DE LA IMAGEN**
Aprovechando el gran avance de los ordenadores personales, es posible conseguir efectos muy vistosos como el encadenado de imágenes.

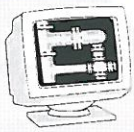
36 **CÓMO HACER UNA DEMO**
Una nueva entrega de esta apasionante serie en la que se explica otro efecto muy común en el mundo de las demos, el plasma.

40 **TÉCNICAS DE SONIDO**
Una vez explicada la reproducción de ficheros de sonido en mono y a 8 bits, es el momento de dar el salto al estéreo con los 16 bits.



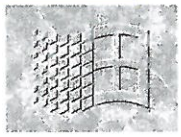
SUMARIO

45 **HERRAMIENTAS DE PROGRAMACIÓN**
Una de las carencias principales de la programación bajo DOS es la falta de concurrencia. Por fin una herramienta salva tal situación mediante THREADS.

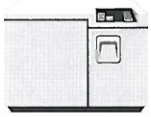


48 **CURSO DE PROGRAMACIÓN EN 3D**
El presente artículo ilustra las posibilidades de realizar simulaciones virtuales 3D. Para ello se explican los métodos matemáticos y gráficos.

53 **CURSO DE PROGRAMACIÓN BAJO WINDOWS**
Uno de los efectos que proporciona mayor profesionalidad a las aplicaciones es la posibilidad de almacenar una configuración personalizada.



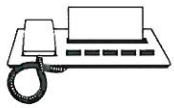
58 **GRANDES SISTEMAS**
La presente entrega de este cursillo pretende introducir al programador en la forma de crear menús para la navegación por el sistema.



62 **SISTEMA OPERATIVO LINUX**
Este artículo realiza una primera introducción al protocolo de comunicaciones TCP/IP, el protocolo más utilizado en los entornos Unix.

68 **PROGRAMACIÓN EN CLIPPER**
Esta nueva entrega del curso explica todas las posibilidades de operación con bases de datos incluyendo las operaciones con índices, filtros y relaciones.

74 **LISTADO BBS**
Se incluye en este número un listado de las BBS por provincias que suponemos le será de mucha utilidad a la mayoría para obtener el software más actual.



75 **PROGRAMACIÓN EN C++**
En la presente entrega de este curso se explica de manera pormenorizada la gestión de eventos para hacer las ventanas completamente operativas.

79 **CONTENIDO DEL CD-ROM**
La versión 1.3.0 de Velocis, junto con una colección de programas shareware y el programa de comunicaciones Littlevt.exe.

81 **CORREO DEL LECTOR**
Este es el espacio dedicado a la resolución de los problemas surgidos a nuestros lectores en los diversos aspectos de la programación.

1er Torneo de Demos e Infografía

Como ya anunciamos en el número de agosto, Sólo Programadores organiza un concurso para los más hábiles y creativos de la programación. Se trata de un campeonato de **Demos** e **Infografía**. Dentro del apartado de demos se incluye también el de **Intros** (como las demos pero más pequeñas). Además del apartado de Infografía se encuentran el de **Vídeo** y el de **Morphing**, de los cuales se habla más adelante.

Existe un suculento premio de **250.000 pesetas en metálico** para los ganadores además de numerosos paquetes de software.

El torneo abrió sus puertas a los más intrépidos el pasado mes de agosto. Durante varios números recogeremos los trabajos realizados por todos aquellos que se animen a participar.

BASES DEL CONCURSO

- Para concursar es imprescindible rellenar y enviar la tarjeta de participación que se adjunta en estas páginas.
- El jurado estará compuesto por la redacción de Sólo Programadores.
- Se podrán enviar uno o varios trabajos en cualquiera de las categorías, que son las siguientes:

- DEMOS:

- No entrarán en el concurso aquellas demos que hayan obtenido primeros puestos en campeonatos internacionales. Sin embargo se podrán enviar para incluirlas en el CD-ROM en el que se publiquen los resultados del concurso.
- Existen varias ramificaciones en la categoría de demos:

INTROS

Son demos de longitud mucho más reducida. Habrá dos grupos: de tamaño menor o igual a **4 Kb** (4.096 bytes) y hasta **64 Kb** (65.536 bytes). Tradicionalmente las primeras no suelen incluir sonido, mientras que las segundas suelen aportar alguna melodía o efecto sonoro.

Las longitudes se refieren al tamaño del fichero ejecutable (que vemos desde el DOS). El archivo .EXE o .COM puede estar empaquetado con la utilidad PK-lite. Dicha utilidad comprime un archivo EXE y genera otro EXE de menor tamaño. Este último cuando se ejecuta se descomprime en RAM en tiempo real quedándose con su tamaño original que se supone es superior. Así Intros que ocupan 100 Kb se reducen a 60 Kb con la utilidad PK-Lite o similar, y pueden entrar en la categoría de 64 Kb.

DEMOS

Se entiende por longitud de la demo la suma de las longitudes de los ficheros necesarios para la ejecución de la demo, bien sea un sólo archivo EXE o uno ejecutable más otros de datos. Dichos ficheros también pueden utilizar técnicas de autodescompresión en memoria como las comentadas en el apartado anterior.

Las demos tendrán un tamaño superior a 64 Kb. La única limitación es la de 4 Mb de memoria RAM total para ejecutar la demo. Se pueden programar demos con extensores de DOS como el DOS4GW de Watcom puesto que el ordenador de prueba será un **486 DX2-66**. Dichos extensores deberán ser enviados en el disquete para poder evaluar la demo.

No se contabilizará como válida la longitud del archivo si está comprimida con uno de los compresores habituales (ARJ, LZH, ZIP, etc.).

- Se enviarán dos copias de la demo. Esto se puede hacer de varias formas. Una consiste en copiar el archivo de la demo dos veces en el disquete (con distinto nombre). Otra forma es enviar dos discos con una copia del archivo en cada uno.

Si es necesario la demo estará comprimida, y si supera la cantidad de 1,4 Mb se podrá empaquetar por volúmenes. **Sólo se admitirán las siguientes compresiones: ARJ, ZIP y LZH.**

En los disquetes irá una pegatina como la que aparece en la **figura 2**. En primer lugar irá el "handle" o nombre del participante/grupo para el concurso y en las casillas se pondrá una X indicando las categorías a las que pertenecen los trabajos enviados en dichos disquetes. Por ejemplo, si un grupo de amigos envía una Intro de 4 Kb en un disquete, pondrá en la pegatina el nombre del grupo de programación y una X en DEMOS, porque la categoría INTROS pertenece a DEMOS.

- INFOGRAFIA:

En las dos categorías, **Imagen** y **Animación**, se admitirán solamente trabajos realizados con programas de infografía o imagen sintética como pueden ser POV, Topas, 3D Studio, etc.).

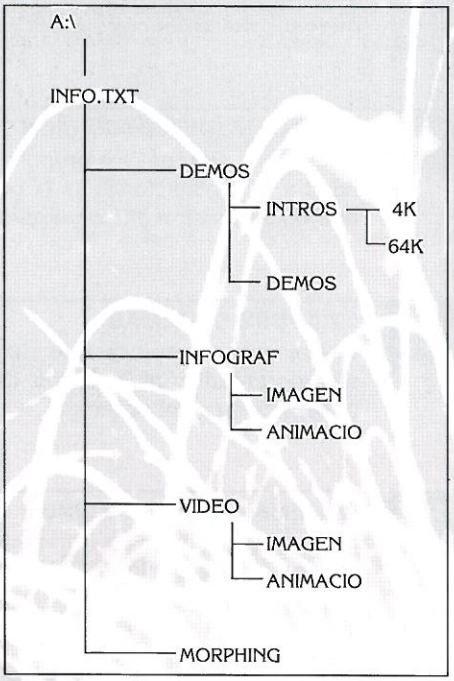


Figura 1.

- **Imagen**. Aquí habrá sub-categorías para englobar los trabajos en función de la resolución y número de colores.
  Sólo serán válidos los trabajos en formato TIF, GIF, JPG y TGA.
  Además de incluir la infografía final se adjuntará obligatoriamente al menos una pantalla del trabajo en fase de construcción. Esto se hará para certificar que el trabajo está siendo realizado por su correspondiente autor.
- **Animación**. En el apartado de infografía en movimiento los trabajos serán admitidos en cualquier formato siempre y cuando se incluya el programa (DOS o Windows solamente) que visualize la animación en un PC 486-DX con 8 Mb RAM y tarjeta True Color VESA.

- VIDEO:

Aquí hay dos grupos similares a la categoría anterior: **Imagen** y **Animación**.

- **Imagen**. Es para los trabajos o composiciones fotográficas en los que se hallan empleado programas específicos para el fotomontaje (Photoshop, Picture Publisher, Photostyler, etc.). Se admitirán sólamente los formatos
  TIF, GIF, JPG y TGA.

(HANDLE O NOMBRE DE PARTICIPACIÓN)

DEMOS [ ] INFOGRAFIA [ ]

VIDEO [ ] MORPHING [ ]

Figura 2.

- **Animación.** Aquí se engloban los trabajos que hayan sido digitalizados como vídeos caseros de humor o imágenes con cierto tratamiento digital (efectos de cortina, fades, etc.). Se admitirán solamente trabajos en formato AVI (Video for Windows) y MOV (QuickTime).

- MORPHING:

Se entiende por trabajos de morphing aquellos en los que a través de una secuencia de vídeo se observan transformaciones de objetos, animales o personas. Para estas secuencias no se permite utilizar personajes del ámbito público (políticos, actores, etc.). Sólo se admitirán ficheros AVI, MOV, FLI, FLC y programas que hagan Morphing en tiempo real siempre y cuando hayan sido programados con rutinas propias de los concursantes.

SOBRE EL MATERIAL ENVIADO

Sólo se admitirá el soporte de disquete y cintas de streamer compatibles QIC-80 para enviar los trabajos. Si se desea que se devuelvan los disquetes y las cintas de streamer hay que adjuntar los correspondientes sobres ya franqueados para que Sólo Programadores realice la devolución.

Los disquetes o las cintas donde se envíen los trabajos tendrán una estructura de directorios como la que se muestra en la **figura 1**.

En cada subdirectorio se colocarán el/los programa/s que vayan a participar. En el fichero INFO.TXT se escribirá en formato ASCII el nombre y teléfono de contacto del representante, así como una breve descripción de cada uno de los trabajos que se envíen.

LOS PREMIOS

Se le darán **250.000 pesetas en metálico** a la mejor y más espectacular **Demo o Infografía**. Los finalistas de las otras categorías recibirán numerosos programas de regalo.

TARJETA DE PARTICIPACIÓN

Nombre:.....Apellidos:.....Edad: .....

Domicilio:.....Cód.postal:.....Teléfono:.....

Pseudónimo o handle para el concurso: .....

CATEGORÍA DE PARTICIPACIÓN

↓ (En cada modalidad se pondrá el número y los nombres de los trabajos enviados) ↓

- Demos [ ]: _____
- Infografía (ficheros .GIF, .TGA, .TIF, FLI, FLC, etc.)
  - Imagen fija [ ]:_____
  - Animaciones [ ]: _____
- Vídeo digital (ficheros .GIF, .TGA, .TIF, .AVI, .MOV, etc.)
  - Imagen fija [ ]: _____
  - Animaciones [ ]: _____
- Morphing [ ]: _____

NORMATIVA PARA EL CONCURSO

- Los trabajos enviados podrán estar realizados por una o varias personas. Si se trata de varias personas solamente el representante aparecerá en la tarjeta de participación. Los datos personales de los otros componentes del grupo serán adjuntados en un folio aparte escritos a máquina.
- No es necesario incluir los fuentes de los programas enviados.
- La responsabilidad de los trabajos enviados atañe tanto al representante que aparece en la tarjeta de participación como a los otros componentes del grupo si es que los hay. Ambos tendrían que responder ante la ley si terceras personas reclamasen la autoría del material o trabajos enviados para este concurso.
- La propiedad intelectual y los derechos de explotación del material enviado pertenecen tanto al representante del grupo como a los demás componentes si los hay. Con la participación en este concurso se concede a la editorial Tower Communications el permiso para publicar en CD-ROM el software enviado para dicho concurso.
- La redacción de Sólo Programadores se reserva el derecho de descalificar a algún participante por un mal funcionamiento del programa enviado o porque dicho programa contenga algún virus informático. También quedarán descalificados aquellos trabajos cuyo contenido sea de mal gusto.
- Enviar esta tarjeta de participación implica aceptar las bases del concurso y la normativa que se estipula en estos apartados.
- La participación en el concurso no será válida si falta algún dato, si hay información incorrecta o bien si la tarjeta de participación es una fotocopia y no la original que viene en la revista. Tampoco será válida la participación en el concurso si la tarjeta enviada no está firmada por el representante.
- La revista Sólo Programadores se reserva el derecho de modificar o ampliar las bases del concurso sobre la marcha si fuese necesario.

Acepto, en representación propia o del grupo, las bases del concurso que aparecen en el número 12 de Sólo Programadores.

Firma del representante:

NOVELL AMPLIARÁ SU OFERTA ACTUAL DE PROGRAMAS PARA DESARROLLADORES

Novell Inc. ha anunciado en su conferencia técnica anual Brainshare la creación del primer programa del sector informático para las relaciones con los desarrolladores. El objetivo perseguido por la compañía es el de satisfacer mejor las necesidades de los desarrolladores de software, integradores, desarrolladores internos y distribuidores de sistemas.

Novell ha adoptado una política de aumento de los mecanismos de comunicación con sus desarrolladores, como ha demostrado con su programa de investigación Dataquest, terminado recientemente.

La presentación del nuevo programa se realizará mediante anuncios periódicos en el transcurso del presente año.

Para más información:
Novell/WordPerfect Spain
Daniel Toledano
Tel: (91) 577 49 41

AST COMPUTER FIRMA UN ACUERDO CON NOVELL

AST Computer acaba de anunciar la firma de un acuerdo a nivel mundial con Novell Inc., cuyo objetivo es ofrecer una integración total de hardware y software a los usuarios de servidores de AST Computer. El acuerdo culminará con la creación de una serie de productos que ofrecerán mejores prestaciones, mayor facilidad de instalación y una total compatibilidad e interoperabilidad. De este modo se ha visto reforzada la relación existente entre ambas compañías en el área de soporte de producto y en los esfuerzos conjuntos de desarrollos de ingeniería.

El acuerdo tiene prevista la licencia de los sistemas operativos de red Netware 3.1x y 4.1 de Novell para AST Computer, quien preinstalará en sus plataformas de servidor dichos sistemas operativos. Asimismo, el acuerdo permitirá que más adelante AST Computer pueda acceder a otros productos de Novell.

Para más información:
Marcos Carmena / Ludi García
Spanish Executive Information Service (S.E.I.S.)
Tel: (91) 308 43 15
Fax: (91) 310 38 76

OLICOM PROPORCIONARÁ SOPORTE PCI TOKEN-RING A LA PERSONAL COMPUTER POWER SERIES DE IBM

Olicom ha anunciado recientemente que proporcionará tarjetas de interfaz de red Token-Ring, basadas en PCI, para la Personal Computer Series de IBM basada en el microprocesador PowerPC.

El adaptador Token-Ring PCI de Olicom proporciona prestaciones óptimas para las aplicaciones más exigentes como puede ser la multimedia. Incorpora un interfaz de bus master PCI de 32 bits y es capaz de alcanzar velocidades de transferencia de datos pico de hasta 133 Mbytes por segundo. Está preparado para los emergentes estándares Token-Ring Full Duplex y respeta la especificación PCI Plug-and-Play. Olicom también ha incluido su tecnología única de driver PowerMACH dentro de los drivers Microsoft Windows NT e IBM AIX, optimizados para el entorno PowerPC.

El adaptador PCI con el soporte de driver PowerPC está disponible en IBM PC Direct como opción en los sistemas Power Series. En el tercer trimestre del año estarán disponibles un software adicional y un kit de documentación para todas las plataformas PowerPC.

Para más información:
Olicom Ibérica
Jacobo de Cal
Tel: (91) 345 72 76
Fax: (91) 457 20 79

SYBASE ANUNCIA LA VERSION BETA 10.5 DE SU SOLUCION OMNICONNECT

Sybase Inc. ha presentado la versión 10.5 beta de su producto

OmniCONNECT, antes conocido como OmniSQL Gateway. El lanzamiento de OmniCONNECT 10.5 constituye el primer paso dentro de una nueva línea de productos interoperativos de segunda generación diseñada para hacer más fácil la vida de los usuarios.

Entre las características de OmniCONNECT 10.5, destacan:

- Un mejor rendimiento y optimización de consultas mediante el optimizador Sybase System 10.
- Soporte para Open Access Servers (OASs).
- Acceso independiente a ORACLE e INGRES, que permite una configuración más flexible y un mejor soporte multitarea.
- Compatibilidad absoluta con Sybase SQL server System 10.
- Proyección en tabla de los resultados RPC.
- Soporte completo para las aplicaciones System 10 CT-Lib.
- Capacidad de gestión de transacciones mejorada.
- Soporte de RPCs transaccionales.

Para más información:
Marcos Carmena / Ludi García
Spanish Executive Information Service (S.E.I.S.)
Tel: (91) 308 43 15
Fax: (91) 310 38 76

NOVELL LANZA INFORMS 4.1 AL MERCADO

Novell Inc. ha presentado el paquete de formularios y acceso a base de datos Informs 4.1. Esta nueva versión incluye nuevas funciones para la automatización de formularios, soporte de bases de datos adicionales y soporte de los estándares del sector.

Informs ofrece 58 nuevas fórmulas y soporte de macros para aplicaciones cruzadas con el lenguaje macro PerfectScript. Presenta un flujo de trabajo mejorado mediante 20 nuevas funciones y ofrece la nueva funcionalidad de los paquetes de correo basados en MAPI, como GroupWise de Novell. Se ha añadido soporte de Open Document Management API (ODMA), ODBC 2.0 y OLE 2.0. También incorpora una integración con bases de datos, con soporte para bases de datos relacionales, ODBC 2.0, IDAPI/ODAPI, Notes FX y NDS.

Informs 4.1 se compone de los paquetes Designer y Filler. El primero permite al usuario la creación de formularios de forma automática mediante un interfaz gráfico, y Filler es el medio con el que los usuarios rellenan, imprimen y dirigen formularios.

Para más información:
Abanico / Hot Line
Liliane Chinyavong
Tel: (91) 594 43 53

SEAGATE REALIZA INVERSIONES MULTIMILLONARIAS EN ESTADOS UNIDOS Y ASIA

Seagate Technology Inc. está realizando inversiones multimillonarias en Asia y Estados Unidos para mantener su liderazgo en la industria. Mediante estas inversiones Seagate reafirma su estrategia vertical de fabricación y se dispone a hacer frente a la creciente demanda del mercado de unidades de disco.

La compañía ya ha comenzado la construcción de una nueva planta de producción en Singapur. Dicha planta incluirá un sistema de manejo de material totalmente automatizado para asegurar una alta calidad en la producción. Está previsto que la planta comience a funcionar a mediados de 1996.

Las inversiones de Seagate en Fremont y Anaheim (California), están orientadas a mejorar sus actuales equipos para incrementar sensiblemente la producción. Seagate confía en doblar su producción antes del final de 1997.

Para más información:
Seagate Technology
Corporate Communications EMEA
62 bis, avenue André Morizet
F-92643 Boulogne-Billancourt Cedex
Francia
Tel: (+ 33 1) 41 86 10 00
Fax: (+ 33 1) 41 86 10 40

NOVELL PRESENTA WORDPERFECT 6.0 PARA UNIXWARE 2 Y AT&T UNIX SVR4 MP-RAS

Novell ha anunciado la disponibilidad de WordPerfect 6.0 para UNIX preparado para su utilización con UnixWare 2 y AT&T UNIX SVR4 MP-RAS. En estos momentos, WordPerfect para UNIX está disponible en siete plataformas UNIX diferentes, cubriendo toda la gama de plataformas Intel UNIX.

La versión 6.0 de WordPerfect para UNIX incorpora, entre otras características, un interfaz totalmente personalizable y más de 70 plantillas automáticas ExpressDoc. También incorpora tablas, módulos de dibujo y gráficos, la utilidad QuickFinder para la recuperación e indexación de texto y el verificador de estilo y gramática Grammatik 5. Las funciones de impresión de la versión 6.0 incorporan el control de una matriz para fuentes comunes a través de diferentes entornos, dando soporte a más de 1.100 impresoras diferentes.

En el próximo trimestre del año habrá una versión de WordPerfect 6.0 para UNIX para las plataformas Siemens Nixdorf y Silicon Graphics (MIPS ABI). También está prevista la comercialización en 1995 de las versiones internacionales de WordPerfect para cada uno de los entornos soportados por UNIX.

WordPerfect 6.0 para UNIX tiene un precio recomendado de 65.000 pesetas. Las actualizaciones del programa tienen un precio de 17.000 pesetas y las actualizaciones de licencia adicionales están disponibles por 12.000 pesetas. Por último, la actualización competitiva tiene un precio de 19.500 pesetas.

Para más información:
Novell/WordPerfect Spain
Daniel Toledano
Tel: (91) 577 49 41

OFENSIVA BORLAND

Borland International vuelve a la carga con el lanzamiento de nuevos productos al mercado. Ya está disponible el Rad Pack para Delphi, una completa colección de utilidades para el producto estrella de Borland que incluye Turbo Debugger, Resource Workshop, Resource Expert, Visual Solutions Pack, las fuentes de VCL y la guía de referencia de Delphi. También está disponible Delphi con los manuales traducidos al castellano y ya hay nuevas versiones del servidor InterBase de Borland.

Por otro lado, la compañía está preparando el lanzamiento de nuevos productos. Ya existen versiones beta de Visual dBase y del compilador Voyager para Visual dBase. Asimismo, está preparada una versión beta de Delphi 32 bits.

Para más información:
DATABASE DM
Tel: (91)6615010

LIBROS

FICHEROS GRÁFICOS. INTERCONEXIÓN ENTRE APLICACIONES

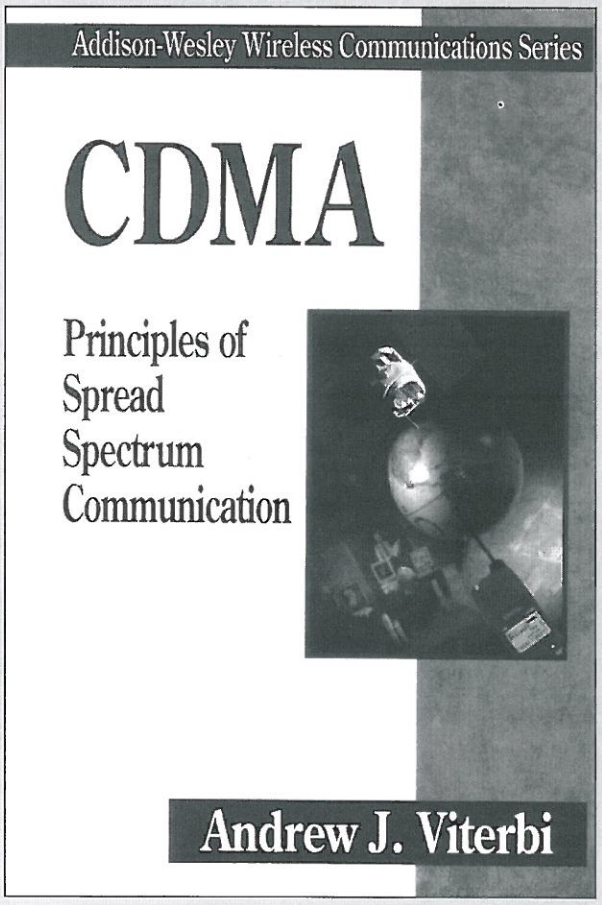
Esta breve obra, perteneciente a la colección "domine al 99%" de la editorial Paraninfo, proporciona al lector los conocimientos necesarios para la utilización de gráficos en sus documentos. El libro comienza con una descripción de los formatos gráficos más usados, sin entrar en detalles, y explica brevemente el intercambio de formatos gráficos. Los demás capítulos están dedicados a la explicación del manejo de gráficos en las aplicaciones ofimáticas más populares, como WordPerfect, Harvard Graphics y Excel. También se habla de programas de dibujo y de utilidades de conversión y de captura, para finalizar con la exposición de algunos ejemplos prácticos.

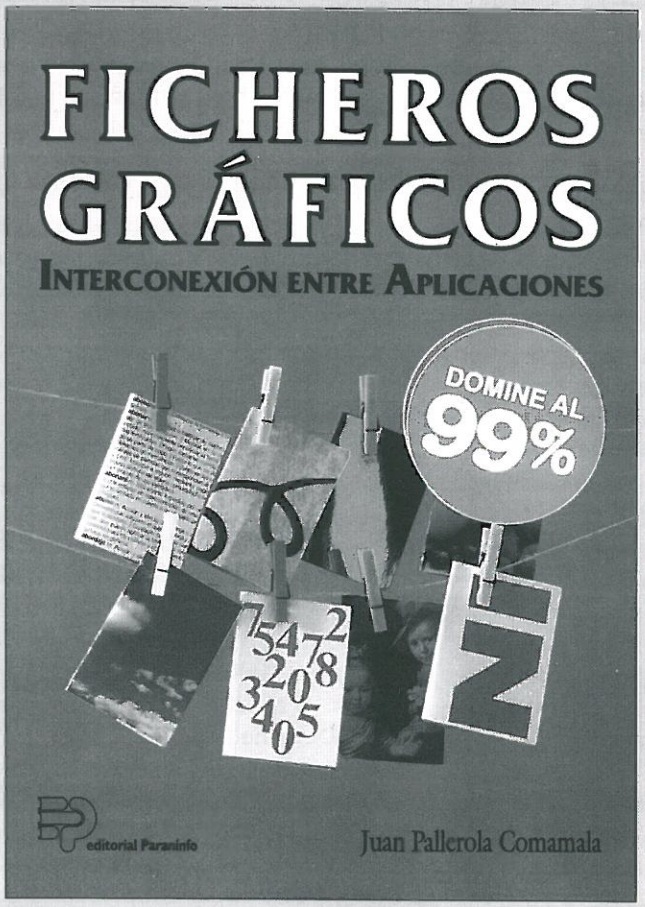
Editorial: Paraninfo
Autores: Juan Pallerola Comamala
81 páginas
Idioma: Castellano
Precio: Consultar

CDMA. PRINCIPLES OF SPREAD SPECTRUM COMMUNICATION

CDMA (Code Division Multiple Access) es una tecnología puntera de comunicaciones, convertida en un estándar por la Telecommunications Industry Association. Esta obra puede servir perfectamente como libro de texto, y está dirigida en exclusiva a los profesionales interesados en la materia. En sus páginas se explican desde los fundamentos de la comunicación digital hasta los más pequeños detalles relacionados con la CDMA. Se trata de una obra rigurosa, bien estructurada y repleta de gráficos aclaratorios que ayudan en la compresión de los complicados conceptos que se exponen.

Editorial: Addison-Wesley
Autor: Andrew J. Viterbi
265 páginas
Idioma: Inglés
Precio: Consultar





¿QUÉ FUE DE LOS PROGRAMAS HAIKU?

Los haiku, poemas japoneses de tres líneas, representan lo que el mundo del software tuvo una vez y quizás haya perdido para siempre. En sólo diecisiete sílabas se las arreglan para comunicarnos una imagen bella y persuadirnos a reflexionar sobre nuestra existencia. Por supuesto, el pseudo-haiku que encabeza estas palabras no pasa de ser una torpe parodia.

Hace años, donde yo estudiaba organizábamos concursos de programas de tres líneas BASIC. Se trataba de destilar lo más elegante de nuestras habilidades y demostrar que un programa diminuto puede tener una potencia desconcertante. Recuerdo que un año, el primer premio fue para un programa que hallaba la gráfica y la integral definida de una función entre dos puntos, y el segundo para un pequeño arcade de plataformas. Todos alucinábamos con lo que se podía conseguir partiendo de una idea sencilla y sometiéndola a un paciente y progresivo refinamiento.

Ni que decir tiene que este espíritu está absolutamente ausente de la escena comercial de hoy. Los juegos y aplicaciones que usamos son cada vez más aparatosos, demandando más y más potencia de nuestras máquinas para las funciones de siempre. Es cierto (a veces) que las prestaciones aumentan, pero el tributo que hay que pagar en incremento de espacio en disco duro y canibalismo de ciclos de reloj es a menudo inaceptable. Se está haciendo buena la siguiente ley informática: "Independientemente de tus recursos económicos, para ejecutar el programa que necesitas es necesario un pecé cien mil pesetas más caro que el tuyo".

Todos los desarrolladores conocemos las razones más obvias de este fenómeno. Por una parte, el uso creciente de esas cosas que ahora llaman "herramientas de desarrollo rápido de aplicaciones (RAD - Rapid Application Development)", y que vienen a ser los 4GL (lenguajes de cuarta generación) de siempre, con la diferencia de que este último término está "out", y ha habido que buscarle un mote más chuli. Con estas herramientas tengo una experiencia curiosa: siempre que las he utilizado ha sido por imposición categórica del cliente, aunque por otra parte lo que el cliente demanda excede claramente las capacidades de la herramienta, con lo cual el ochenta por ciento del tiempo se dilapida en hacer trucos inconfesables para saltarnos las limitaciones del entorno. Con lo cual se consigue: la lentitud y ocupación en disco duro de un 4GL, y la dilatación del tiempo de desarrollo y la poca mantenibilidad de la programación artesana, todo en el mismo paquete. Maravilloso.

Por otra parte, la pandemia de los últimos cinco años: el esnobismo gráfico, que provoca el convencimiento en el cliente y en los gestores de proyecto de que una aplicación de gestión de almacén tiene que parecerse a un corto de continuidad de la MTV. Ya saben. El listado de proveedores, que antiguamente se ejecutaba seleccionando la opción "Listados" del menú principal, y seguidamente, la opción "Proveedores" del menú de listados, ahora se invoca del modo siguiente: Hacer doble clic en el icono del hombre gordo con gorra, que representa al proveedor (no confundir con el de la señora del portafolios, que es la que representa al cliente), esperar a que baile "Me and my shadow", un clásico del claqué, mientras dice amigablemente: "Proveedores" a través de la tarjeta de sonido de 16 bits, esperar a que se despliegue el menú de acciones del proveedor con un sonido como del Talmud desenrollándose con suavidad píxel a píxel, y seleccionar la opción "Ver Todos", con lo que la pantalla se nos llena de sonrientes hombrecillos gordos con gorra, ir a "Editar", "Seleccionar Todos" (o bien seleccionar el primero con un clic del ratón manteniendo pulsada la tecla Mayúsculas, y luego un clic en el último gordito sin soltar dicha tecla), y luego arrastrar al ejército de mofletudos al icono de la impresora. Así es como, gracias al uso mesurado e inteligente de los avanzados entornos gráficos, el software se utiliza cada vez de una forma más intuitiva.

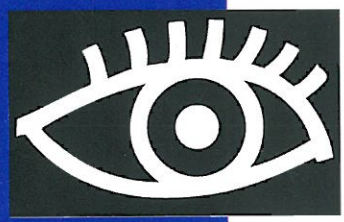
Y finalmente, el culto a las cifras que los que hacemos software hemos sabido inculcar en el público. "¡Diecinueve gigas de información! ¡Más de cincuenta vídeos a todo color! ¡Más de ciento ochenta minutos de sonido estéreo!". Eso es lo que ponemos en las cajas y en los anuncios de nuestras creaciones. Nada de "¡Código optimizado! ¡Funciona incluso en el XT de su abuela Francisca! ¡Sólo tiene las opciones que necesita, pero las hace bien!". No, eso no vende. Y así nos va: tenemos que estrujarnos la mollera escribiendo columnas de opinión en nuestro tiempo libre para comprarnos un segundo disco de un giga y así poder instalar la última versión de nuestro compilador favorito, con el que, por supuesto, podremos escribir programas mastodónticos llenos de opciones que nadie usará jamás. El haiku ha muerto. Viva el poema de Mío Cid.

Casi todo en este disco duro
es agotador y prescindible,
como casi todo en mi vida.

Y-Lu-So, siglo XVII

VISUAL BASIC O DELPHI, LA GRAN PREGUNTA

Luis Caballero



Hasta ahora, cualquier programador que tuviese intención de trabajar de una forma sencilla y rápida para crear aplicaciones Windows tenia que recurrir irremisiblemente al Visual Basic 3.0. Ahora Borland entra también en juego con Delphi, lo que ha generado la gran pregunta sobre cuál utilizar. Este artículo hace una comparación de ambas herramientas.

Cualquier persona que se haya dedicado a programar para Windows con herramientas como Borland C++ o Microsoft C++ sabrá de la dificultad que conlleva esta tarea. La creación del interface, los recursos, etc. suponían una carga sobre la realización de las aplicaciones que en la mayoría de los casos aumentaban al doble o más los esfuerzos y el tiempo de creación de dicha aplicación. Aparte de esto, la creación de aplicaciones Windows quedaba reservada para unos cuantos privilegiados con amplísimos conocimientos sobre programación y Windows, así como del lenguaje de programación utilizado.

Fue a mediados del 93 cuando Microsoft lanzó al mercado de la versión 3.0 de una herramienta denominada Visual Basic. Esta herramienta acercaba la realización de aplicaciones Windows a toda aquella persona con mínimos conocimientos del lenguaje Basic. Debe además tenerse en cuenta que Basic es el lenguaje elegido por la mayoría de programadores para aprender a programar, tanto aquellos que ahora son expertos como los que se introducen en la actualidad.

Debido al lenguaje utilizado (Basic) fueron muchos los programadores que condenaron a esta herramienta al fracaso. Pero con el tiempo descubrirían el grave error que cometieron. Poco a poco Visual Basic comenzó a crear estándares sobre diferentes campos, como la creación de controles, manipulación de Bases de datos, etc. Fue la misma razón que muchos argumentaron para pensar en su fracaso (el lenguaje utilizado), el que contribuyó a su mayor difusión. Personas que jamás habían programado (no sólo para Windows, sino para DOS) eran capaces de crear en un corto período de tiempo aplicaciones con aspecto profesional, mientras a los programadores expertos se les facilitò su trabajo.

La disyuntiva ante la que se encontraron las empresas dedicadas a la creación de aplicaciones Windows fue muy clara, ¿Es preferible aumentar el tamaño de nuestras aplicaciones pero crearlas en la mitad de tiempo?, la respuesta no se hizo esperar. Es mucho más lógico, teniendo en cuenta el estado actual de los recursos de los sistemas informáticos, aumentar el tamaño de una aplicación en unos pocos cientos de Kbytes, pero a cambio, poder entregárselo al cliente en menos de la mitad del tiempo que antes. La consecuencia fue que la mayoría de las empresas adoptaron Visual Basic como su herramienta de trabajo, dejando lenguajes como el C++ para la creación de pequeñas funciones auxiliares para las que esta herramienta no tenía cobertura.

APARECE LA COMPETENCIA

Las empresas de la competencia dejaron transcurrir el tiempo para ver la evolución del Visual Basic, y en función de los resultados de dicha herramienta ponerse manos a la obra en la creación de herramientas similares. Pero, gracias al buen resultado que dio Visual Basic y su gran aceptación entre la comunidad de programadores, ese tiempo se convirtiò en un precioso tiempo perdido por la competencia. Fue a principios de año cuando el histórico competidor de Microsoft en el desarrollo de herramientas de programación, Borland, sacó al mercado la competencia para Visual Basic. La nueva herramienta, denominada Delphi, es lo que coloquialmente puede denominarse "Visual Pascal", es decir, una herramienta gemela a Visual Basic, pero cambiando el lenguaje Basic por Pascal.

Desde ese momento, la comunidad de programadores ha sufrido una pequeña conmoción tratando de intuir cuál de las dos herramientas será la que se lleve el gato al agua.

LA PRIMERA DIFERENCIA, SU FILOSOFÍA

A pesar de las grandes similitudes existentes entre ambas herramientas, cada una de ellas posee ciertas peculiaridades que la diferencian de su rival. Para empezar, Visual Basic es un *pseudocompilador*, ya que las aplicaciones creadas con esta herramienta necesitan de ciertas librerías complementarias para su ejecución. En concreto, las aplicaciones de Visual Basic deben ir acompañadas de la librería VBRUN300.DLL. Esta misma razón le impide a Visual Basic crear librerías de enlace dinámico (DLL) completamente independientes. Sin embargo, Delphi es un compilador, por lo que las aplicaciones creadas con esta herramienta son ficheros ejecutables independientes, es decir, sin necesidad de librerías complementarias. Esta característica le permite además crear librerías de enlace dinámico independientes.

La elección del lenguaje Basic por parte de Microsoft ha contribuido a su extensión. Basic es el lenguaje por excelencia con el que se aprende a programar, debido a la facilidad del mismo. La versión del lenguaje utilizada por el Visual Basic 3.0 es una ampliación del Basic estàndar. La actual versión cuenta con mejoras como son la estructuración completa del lenguaje, la inclusión de nuevos tipos de datos (como los objetos) y funciones específicas para el manejo de bases de datos.

Borland, como ya es costumbre, se decidió sin embargo por la utilización del lenguaje Pascal, aquel que le lanzara a la fama hace ya bastantes años. Tampoco en este caso se trata de la versión más o menos estándar del lenguaje, sino del denominado *Object Pascal*, una versión extendida del lenguaje Pascal en la que se incluyen entre otras la posibilidad de realizar Programación Orientada a Objetos.

EL ENTORNO

Una vez enunciadas las diferencias más grandes, es momento de comenzar a entrar en el detalle. El primer paso en este sentido son las diferencias entre el propio entorno de programación.

Tal y como puede comprobarse en la figura 1, Visual Basic ofrece desde el principio las siguientes ventanas:

- Menú principal: Es la ventana principal de Visual Basic, y en ella se encuentran los menús para poder abrir y cerrar archivos, compilar, etc.
- ToolBox: Es la caja de herramientas, en ella aparecen todos los controles que se encuentran disponibles.
- Form1: Esta es la ventana del formulario, que actuará como ventana principal de la aplicación.
- Properties: Es la ventana de propiedades, en ella se asignan los valores de las propiedades de los controles.
- Project: En esta ventana aparecen todos los ficheros que forman parte del proyecto.

Por último, pueden incorporarse dos ventanas más. La primera, denominada *Code*, es la ventana de código. En ella se introduce el código correspondiente a los eventos y métodos de todos los controles.

Existe además la posibilidad de incluir una nueva ventana denominada *ventana de módulo*. En ella aparece un editor mediante el cual puede introducirse código en Basic complementario a la aplicaciòn.

Delphi por su parte aporta un entorno más compacto. Tal y como puede comprobarse en la figura 2, Delphi presenta menos ventanas, si bien, cada una de ellas representa una herramienta distinta que, a su vez, integran dentro otras ventanas. Además, el entorno Delphi está diseñado en forma de carpeta o archivador. De esta forma, cada ventana cuenta con una pequeña pestaña, mediante la cual puede accederse a la misma. Las ventanas de Delphi son:

- Menú principal: Esta ventana es muy similar al menú principal de Visual Basic. La diferencia entre ambas radica en que dentro de esta ventana se incluye también la caja de herramientas. Ésta tambièn aparece organizada también como una carpeta.
- *ObjetcInspector*: Es una herramienta dedicada al manejo interno de los objetos y controles de la aplicación. Mediante esta herramienta pueden manipularse, por un lado, las propiedades de los objetos, y por el otro los eventos de cada uno de los objetos.
- Form: Es la ventana del formulario y representa la ventana principal de la aplicación.
- Editor de código: Es un editor de texto mediante el cual se introduce el código de la aplicación. El contenido de esta ventana se va incluyendo de forma automàtica.

Delphi también permite incorporar otras ventanas más al entorno, como por ejemplo el *ObjetcBrowser*, una aplicación para representar de forma gráfica la estructura de objetos y las relaciones entre ellos.

TABLA 1

	Visual Basic	Delphi
Programación Visual	Sí	Sí
Lenguaje de alto nivel	BASIC	PASCAL
Controles VBX	Sí	Sí
Acceso a Bases de Datos	Sí	Sí
EXEs independientes	No	Sí
Acceso a funciones de DLLs	Sí	Sí
Creación de DLLs	independientes	Sí
Compilador de Ayudas	Sí	No
Consultas SQL	Sí	Sí
Memoria mínima requerida	2Mb.	4Mb.
Memoria mínima recomendada	4Mb.	8Mb.
Tamaño aproximado en HD	30Mb.	30Mb.
Versión 32 bits	Sí	Beta
Soporte OLE 2.0	Sí	Sí
Soporte DDE	Sí	Sí
Motor de Gestor de Base de datos	MS Access	dBase 5 para Windows
Herramientas complementarias	ODK	RAD Pack
Generador de informes	Crystal Report	Report Smith
Programa para crear instalaciones	Sí	No

Especificaciones técnicas.

LAS DIFERENCIAS EN EL CÓDIGO

Las dos herramientas siguen de manera bastante aproximada la especificación de programación visual, es decir, ambas realizan el diseño del interface con el usuario mediante el ratón, de forma que el programador sólo tendrá que pinchar los controles y situarlos en la pantalla.

Ahora bien, no todo es simplemente introducir controles en la pantalla, sino que además deben configurarse y programar la respuesta adecuada a los eventos.

Como ya se ha mencionado anteriormente, Visual Basic utiliza como lenguaje de programación el Basic. Esta versión del lenguaje se aleja bastante del Basic clásico. Mediante este lenguaje pueden realizarse la mayor parte de las operaciones que un programador necesita para realizar aplicaciones de propósito general. La principal ventaja que ofrece este lenguaje es la facilidad tanto de uso como de aprendizaje. La pega que puede ponérsele es que, aquellos programadores acostumbrados a lenguajes tipo C o C++, encontrarán que Basic proporciona un lenguaje algo más limitado, incluso a pesar de las ampliaciones introducidas en el mismo. A pesar de esta consideración, la potencia de esta versión mejorada del Basic es más que suficiente para la mayoría de los programadores, y puede decirse que sin llegar a dominar por completo el lenguaje pueden realizarse aplicaciones francamente profesionales.

En el otro lado, Delphi utiliza como base el lenguaje Pascal. Este lenguaje es de por sí más potente que el Basic. Además, el Pascal utilizado por Delphi se corresponde con el *Objetc Pascal*, que es la versión del lenguaje utilizado por el Borland Pascal 7.0 para Windows. Dicho lenguaje permite realizar cualquier operación que el programador desee, como la creación de clases nuevas, accesos a bajo nivel, etc. Como casi siempre, el aumento de potencia de cualquier lenguaje provoca un aumento en la dificultad de utilización y aprendizaje del mismo. El propio Pascal, a pesar de ser uno de los lenguajes recomendados para aprender a programar, no puede escapar de esta consecuencia. Pascal es un lenguaje de programación capaz de satisfacer a los programadores más exigentes.

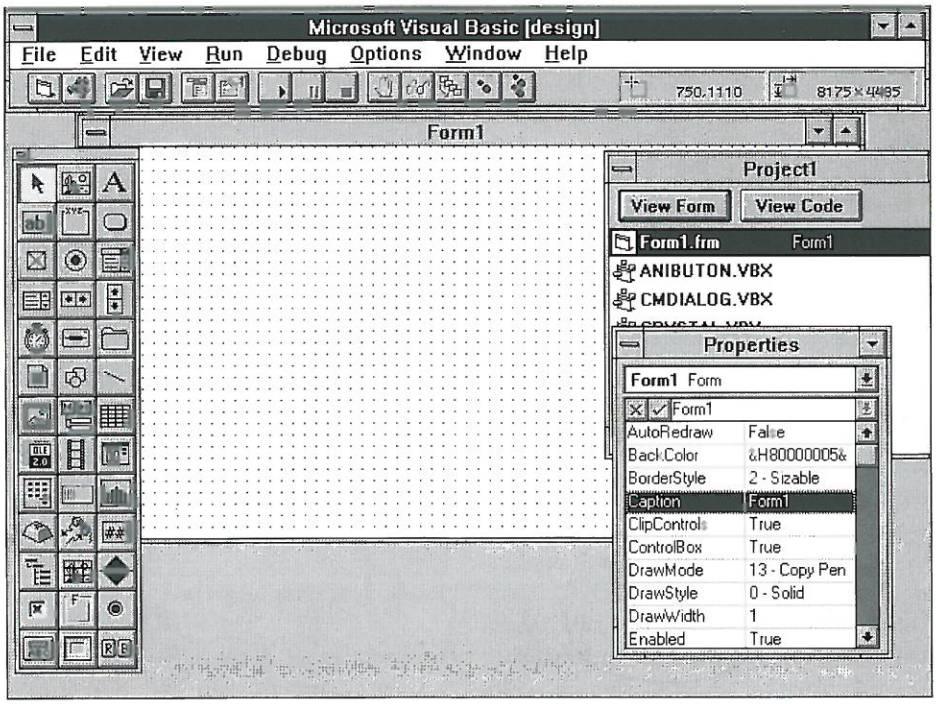


Figura 1. El entorno del Visual Basic 3.0.

Cuando alguien comienza el aprendizaje, Pascal resulta un lenguaje relativamente sencillo, pero en el momento en el que se pretende profundizar en cuestiones como Programación Orientada a Objetos o el manejo dinámico de la memoria, la dificultad en el aprendizaje aumenta bastante. Esto no es consecuencia del lenguaje, sino de la propia filosofía de programación.

En cuanto al lenguaje utilizado por ambas herramientas, no cabe la menor duda que es mejor el Pascal de Delphi que el Basic de Visual Basic. A pesar de llo, en líneas generales, Delphi requiere del programador bastantes más conocimientos de programación para realizar una aplicación que Visual Basic.

HERRAMIENTAS DE TERCEROS

Otra de las cuestiones a tener en cuenta son las posibilidades de expansión que ofrece el lenguaje, así como la posibilidad de conexión de las aplicaciones desarrolladas con cualquiera de las dos herramientas y las aplicaciones de terceras partes. En este sentido deben diferenciarse varios apartados.

El primero es la posibilidad de enlace y acceso a bases de datos. Para ello, ambas herramientas aportan varios controles que permiten la manipulación de los datos. Hasta aquí, todo parece muy similar, pero en realidad aparecen grandes diferencias.

Visual Basic aporta dos formas básicas de acceder a las bases de datos. El primero de estos métodos es mediante el control *DataControl*, por el cual puede recorrerse la base de datos registro a registro, y realizar cualquier tipo de consulta. Sin embargo, no permite hacer operaciones de consulta avanzadas. Para ello, Visual Basic aporta el *Data Manager*. Se trata de una utilidad que funciona de la misma forma que un gestor de bases de datos, es decir, utiliza un modo funcional similar al *Access*. Esto supone, por un lado, la ventaja de que se trata de una potente herramienta con la que el programador ya cuenta, pero como contrapartida exige ciertos conocimientos previos de los que vayan a ser usuarios finales.

Delphi, sin embargo, proporciona diferentes controles para cada una de las posibles operaciones a realizar con una base de datos, como por ejemplo, controles para consultas SQL, edición y visualización de campos, etc.

Esto proporciona sin duda alguna una mayor flexibilidad para crear aplicaciones, pues disponemos de controles mucho más concretos. Por otro lado, si se desea realizar una aplicación más profesional, esta circunstancia va a

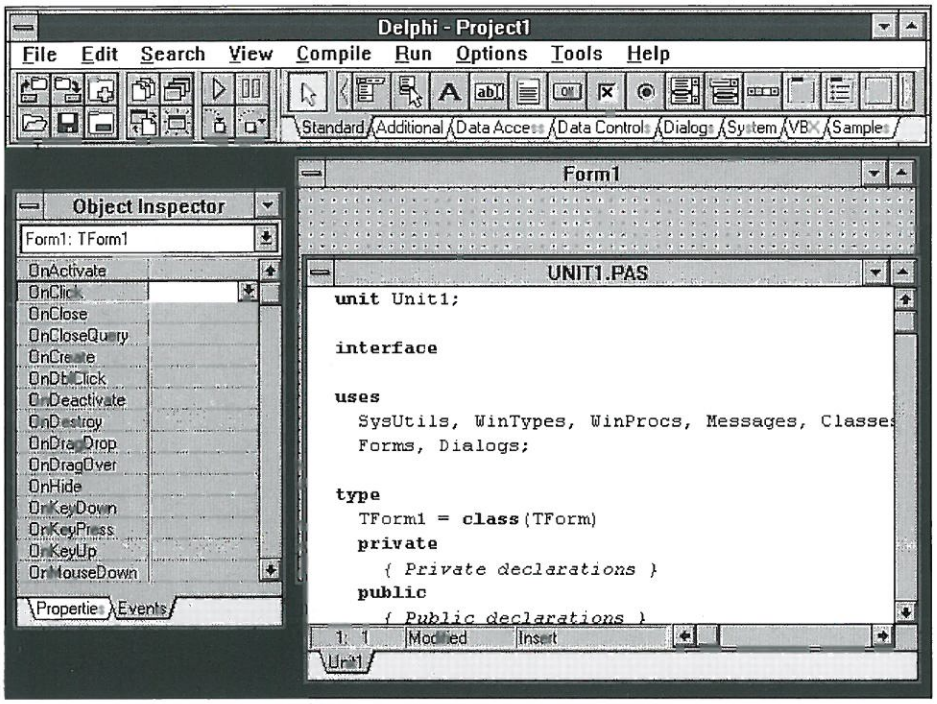


Figura 2. El entorno de Delphi.

suponer una mayor cantidad de trabajo.

Además, cada una de las herramientas cuenta con las librerías del gestor de bases de datos de sus respectivos fabricantes. En este sentido, Visual Basic proporciona las librerías y el motor utilizado por el *Microsoft Access*, mientras que Delphi aporta lo mismo, pero en este caso referido a *dBASE 5 para Windows*.

Otro de los aspectos importantes son las ampliaciones de librerías, clases, etc. En este sentido, Visual Basic se impone sin lugar a dudas, ya que es el creador de un estándar en cuanto a controles personalizados. Este estándar recibe el nombre de *VBX* (*Visual Basic eXtensions*), y se trata de controles realizados por empresas o particulares, que pueden ser utilizados por cualquier aplicación realizada con Visual Basic. Delphi también es capaz de reconocer este tipo de controles y utilizarlos en sus aplicaciones. Los controles de este tipo se distribuyen en ficheros con la extensión .VBX, y además suelen contar con librerías DLL que contienen funciones de apoyo al control. La inclusión de estas librerías no supone ningún problema, pues ambas herramientas son capaces de utilizar funciones incluidas en ellas sin ningún problema.

Y ADEMÁS DEL COMPILADOR ¿QUÉ?

Otro de los factores importantes son aquellas utilidades que acompañan a cada uno de los programas en sus respectivos paquetes. Éstas generalmente van encaminadas a la realización de partes concretas de la aplicación.

Visual Basic proporciona una serie de herramientas dentro del mismo paquete que son de gran interés para el programador.

- *Help Compiler*: Compilador de ayudas para Windows.
- *Crystal Report*: Es una utilidad para la elaboración de informes con el contenido de los registros de bases de datos.
- *Setup Wizard*: Una aplicación capaz de realizar ficheros *Setup* para la instalación automática del programa creado con Visual Basic en su distribución.

El paquete de Delphi incorpora algunas utilidades similares a las de Visual Basic:

- *Report Smith*: Un programa para la elaboración de informes similar al *Crystal Report*.
- *InterBase*: Un servidor local compatible ANSI SQL-92.

PAQUETES COMPLEMENTARIOS

Tanto Microsoft como Borland cuentan con paquetes complementarios a sus respectivas herramientas. Estos paquetes contienen controles adicionales, manuales más extensos, y una serie de ayudas que el programador puede echar de menos en los paquetes de distribución de las herramientas.

En lo concerniente a Visual Basic, Microsoft proporciona el denominado *ODK (Office Developer's Kit)*. En él se incluyen ciertas utilidades complementarias como guías de programación y diseño integradas, referencia completa sobre la especificación OLE 2.0, ayudas para la creación de aplicaciones como Excel o Word, información técnica sobre Microsoft Access, y controles adicionales para la realización de aplicaciones para tratamiento de correo electrónico.

Mientras, Borland ha sacado al mercado el *Radical Pack for Delphi*. Se trata de una serie de utilidades complementarias encaminadas a mejorar las prestaciones del propio Delphi. Estas utilidades consisten en las guías de referencia del lenguaje, el código fuente de los componentes visuales (*VCL*), el *Resource WorkShop* y el *Resource Expert*, dos editores de recursos para el diseño de menús, cajas de diálogo, iconos, etc. Por último, el paquete incorpora el *Visual Solution Pack*, compuesto por un conjunto de controles para diferentes utilidades como sonido, animaciones, comunicaciones, editores, etc.

EL FUTURO

Resulta evidente que estas dos herramientas son la avanzadilla o cabeza visible de las futuras herramientas de programación. Ambas apuntan hacia un futuro en el cual el programador clásico va a quedar relegado a unas pocas funciones, como las de crear utilidades y librerías complementarias a estas herramientas.

Cada una de las dos herramientas comienzan ya a trabajar en el futuro. Visual Basic tiene anunciado ya el lanzamiento de la versión 4.0 para el próximo mes de octubre, estando disponible ya la versión Beta para su evaluación. Además, en el mes de diciembre aparecerá una versión ampliada de esta herramienta encaminada al desarrollo de aplicaciones Cliente/Servidor.

Borland por su parte parece volcarse con Delphi y, además de la aparición del mencionado *Radical Pack*, se está ultimando una versión de 32 bits para realizar aplicaciones para Windows 95 y

Windows NT. En la actualidad existe una versión Beta para su evaluación, y hace pocos meses fue presentada una versión preliminar por el propio Zack Urlocker, uno de los autores del Delphi y toda una figura dentro del lenguaje Pascal. En dicha presentación se aseguró que la versión definitiva de este compilador saldrá pocos meses después del lanzamiento mundial de Windows 95. Además, se trata de un compilador completamente nuevo, y no como es habitual, una mejora del ya existente de 16 bits. Entre los avances que se anuncian con esta herramienta cabe destacar una total compatibilidad con la versión de 32 bits, y un perfecto funcionamiento en entornos de 32 bits.

Por último, Borland también ha anunciado la próxima salida al mercado de un conversor de código capaz de traducir el código de una aplicación creada con Visual Basic a Delphi. En la actualidad existe una herramienta que realiza dicha operación, pero debido a que es ajena al propio Borland, sólo es capaz de traducir aproximadamente el ochenta por ciento del código.

OTRAS HERRAMIENTAS VISUALES

Tal vez lo que más se eche de menos, a la vez que el mayor reto con el que se encuentran ambas casas, sea la posibilidad de contar con una herramienta tan fácil de manejar como éstas, que permita el trabajo con el lenguaje C++. Es de suponer que numerosos lectores pensarán que existe una herramienta como tal, el Visual C++, pero que nadie se llame a engaño porque esto no es así. Visual C++ no es una herramienta visual cien por ciento. Esta herramienta se halla a medio camino entre la programación clásica y la programación visual. Supone un avance importante, pero todavía no puede comparase con las dos herramientas analizadas.

Borland por su parte distribuye el denominado *Visual Solutions Pack*, una herramienta que se aproxima al diseño visual pero utilizable únicamente para recursos.

CONCLUSIÓN

Una vez analizadas las dos herramientas pueden sacarse varias conclusiones claras. A pesar de todo, este artículo no tiene intención de hacer una crítica a una u otra herramienta, ya que ambas son realmente buenas. La verdadera y última intención de este artículo es presentar al lector las dos herramientas más populares para la creación de aplicaciones Windows de una forma visual, y exponer las diferencias que presentan ambas.

Una vez expuestas las similitudes y diferencias de las herramientas de programación más de moda en la actualidad, llega el momento de las valoraciones. La verdad es que resulta realmente difícil decantarse por una o por otra, y debe hacerlo el propio programador. Si se poseen pocos conocimientos de programación, la opción más ventajosa es la proporcionada por Visual Basic. Si por el contrario se poseen amplios conocimientos de programación o se desea poseer un control exhaustivo sobre todos los procesos de la aplicación, el lector debe decidirse por la opción Delphi.

Aparte de los conocimientos informáticos, también debe tenerse en cuenta el equipo con el que se cuente. Las exigencias de procesador y memoria varían bastante de una a otra herramienta, como el lector podrá observar en el cuadro 1. Los requisitos para poder desarrollar con normalidad (al margen de los exigidos y recomendados por los propios fabricantes) son:

- Visual Basic puede ejecutarse con completa normalidad en un ordenador 486DX a 33 o 50 MHz. con 4 Mbytes de memoria. El espacio ocupado en disco duro es de unos 30 Mbytes.
- Delphi se ejecuta con normalidad en ordenadores 486DX2 a 66 MHz. o superiores con un mínimo de 8 Mbytes de memoria. La ocupación en disco duro es también de unos 30 Mbytes.

A los mandos de las herramientas

Cuando se comienza a programar con cualquiera de las dos herramientas, se percibe lo sencillo que puede ser el desarrollo de aplicaciones. Todo basado en un interface gráfico, en el cual únicamente debe hacerse click y arrastrar controles dentro de los formularios.

Sin embargo, las primeras diferencias en ejecución comienzan con la velocidad a la que arrancan las propias herramientas. Visual Basic es algo más rápido a la hora de arrancar que Delphi. La diferencia depende sobre todo de la memoria con que cuente el ordenador, y así, cuanto mayor sea la memoria, menor será la diferencia. Por ejemplo, el arranque en un ordenador con 4 Mbytes de memoria de Delphi supone casi un minuto, mientras que Visual Basic sólo consume poco más de 15 segundos. Si el ordenador cuenta con 16 Mbytes, la diferencia es mínima, pues Delphi consume 15 segundos aproximadamente, mientras que Visual Basic poco menos de 10 segundos.

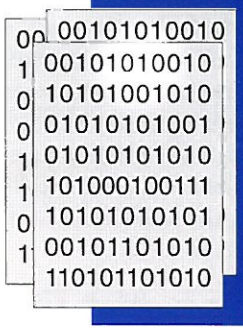
A la hora de realizar la ejecución de un programa desde el propio entorno con el fin de probarlo, hay que tener en cuenta ciertas cuestiones. La primera vez que se realiza esta operación, Delphi tarda algo más que Visual Basic. Sin embargo, Delphi almacena compilada la parte que no varía de los programas, por lo que en las sucesivas ocasiones en que se realice esta operación (dentro de la misma sesión de trabajo), el tiempo consumido se reduce considerablemente, incluso menor que el de Visual Basic. Visual Basic, por su parte, se mantiene estable, pues dado que es un pseudocompilador necesita compilar menos líneas.

Los tiempos de realización de un fichero ejecutable son muy parecidos, ya que, a pesar de que Visual Basic tiene que traducir menos líneas de código, el compilador aportado por Delphi es el más rápido del mercado. Resulta evidente que además estos tiempos irán en función del procesador que tenga el ordenador, así como la cantidad de memoria del mismo.

Otra de las posibles comparaciones es el tamaño final de la aplicación una vez compilada. En este sentido deben tenerse en cuenta también ciertas consideraciones. Debido a la naturaleza de Visual Basic, las aplicaciones generan un fichero EXE relativamente pequeño. Sin embargo, esto no debe llevarnos a engaño, pues para su ejecución necesita de la librería VBRUN30.DLL. Dicha librería tiene un tamaño próximo a los 400 Kbytes. Mientras, los ficheros de Delphi rondan un mínimo de 200 Kbytes. Sin embargo, debe tenerse en cuenta que la librería VBRUN300.DLL puede ser compartida por todas las aplicaciones realizadas con Visual Basic. De esta forma, cuando el programador se junta con tres o más aplicaciones realizadas con la misma herramienta, el espacio ocupado por las mismas será menor si son de Visual Basic que si son de Delphi.

Por último, resulta también importante comparar los tiempos de ejecución de las aplicaciones realizadas con dichas herramientas. La velocidad es sensiblemente mayor cuando se ejecutan aplicaciones realizadas con Delphi que las realizadas con Visual Basic.

En resumen, la principal diferencia entre ambas radica en la forma en que se ven afectadas por el procesador del ordenador y por la memoria del mismo. Así, si el ordenador cuenta con un procesador 386 o 486DX, y con una memoria de entre 4 y 8 Mbytes, la diferencia en tiempos de ejecución es bastante favorable al Visual Basic, mientras que si se trata de un ordenador con un procesador 486DX66, OverDrive o Pentium, con más de 8 Mbytes de memoria, los tiempos se equiparan, por lo que resulta mejor el Delphi. En cuanto al espacio, ambas herramientas ocupan en disco duro una cantidad de espacio muy similar, y las aplicaciones depende del número de éstas. Para una o dos aplicaciones, Delphi proporciona un espacio menor, mientras que en el momento en el que se alcancen las tres o más aplicaciones, el espacio de las aplicaciones realizadas con Delphi comienza a dispararse, mientras que con Visual Basic tiene un crecimiento menos brusco.



INTRODUCCIÓN A LAS MACROS

Santos Riancho

Cuando se codifica un programa es frecuente que dentro del mismo deban ejecutarse varias veces las mismas instrucciones. En los capítulos 6 y 7 se vio la manera de reducir el tiempo de programación y el tamaño del ejecutable "encapsulando", dentro de procedimientos, esas instrucciones mediante las directivas PROC y ENDP.

Sin embargo, no siempre que un fragmento de código aparece varias veces en un programa resulta adecuado darle forma de procedimiento. Piénsese, por ejemplo, en algo tan habitual como definir la posición del cursor en la pantalla. Esta labor puede ser llevada a cabo con ayuda de la función 02h de la interrupción 10h (gestor de vídeo del B.I.O.S.), que requiere que se deposite en los registros DH, DL y BH los valores correspondientes a la fila, columna y página de vídeo, respectivamente, donde será situado el cursor. La ejecución de un procedimiento que, basándose en lo anterior, situara el cursor en un punto de la pantalla implicaría, por lo tanto, pasar a la pila tres valores de un byte, ejecutar la instrucción CALL y las que componen la rutina y retornar al módulo principal.

Basta con plantearse sólo el paso de parámetros de este hipotético procedimiento para llegar a la conclusión de que su uso no optimizaría el tamaño del archivo fuente ni el del ejecutable; más aún: el programa resultaría relentizado debido al tiempo consumido en la ejecución de, entre otras, las instrucciones de llamada y retorno. En estas circunstancias la mejor solución no es otra que escribir el código encargado de posicionar el cursor tantas veces como sea necesario.

FACILITAR LA ESCRITURA DEL FUENTE

Este trabajo se ve notablemente simplificado con el uso de las *macros*. Una macro es un nombre simbólico al que están asociadas una o más instrucciones. Una vez que se ha definido la macro, basta con invocarla desde cualquier punto del archivo fuente para que el programa ensamblador inserte las instrucciones de que consta, como si el programador las hubiera tecleado. A esto es a lo que se llama *expansión de la macro*.

Las macros suelen ser definidas al comienzo del archivo fuente y, para ello, son necesarias las directivas *MACRO* y *ENDM*, que se emplean de la forma mostrada a continuación:

```
nombre MACRO
  ...
  <instrucciones>
  ...
ENDM
```

Un ejemplo de utilización es la construcción de la macro *Parar*, que introduce una pausa en el programa en el punto en que se escribe dicha palabra.

```
Parar MACRO
  mov ah, 00h
  int 16h
ENDM
```

Como es posible deducir a partir de este caso, el empleo de macros agiliza la codificación, ya que, por un lado, evita teclear una y otra vez la misma secuencia de instrucciones y, por otro, aumenta la legibilidad del archivo fuente. Estas características son propias también de los procedimientos, pero no

Las macros son elementos muy importantes dentro de un programa ensamblador ya que, con su ayuda, se pueden agilizar notablemente tareas para las que un procedimiento resultaría inadecuado. El presente capítulo es una introducción a sus posibilidades.

olvidemos la gran diferencia entre ambos elementos del lenguaje: mientras que un procedimiento aparece una sola vez dentro del ejecutable, el código asociado a una macro es insertado en el programa tantas veces como ésta es invocada dentro del mismo.

MACROS, PARÁMETROS Y ARGUMENTOS

De todas formas, las macros serían de poca utilidad si sólo pudieran ser reemplazadas por una única secuencia de instrucciones. Por ello el lenguaje ensamblador ofrece distintos recursos para hacer más elástica la generación de código en la expansión de una macro. El primero que vamos a contemplar es el uso de parámetros, que se ilustrará con el caso expuesto al principio: el posicionamiento del cursor en un punto de la pantalla.

FIGURA 1

```
PonCursor MACRO Pagina, Fila, Columna
  mov ah, 02h
  mov bh, Pagina
  mov dh, Fila
  mov dl, Columna
  int 10h
ENDM
```

FIGURA 2

```
PonCursor MACRO Pagina:=<00h>, Fila, Columna
```

FIGURA 3

```
PonCursor MACRO Pagina:=<00h>, Fila:REQ,
Columna:REQ
```

Como se dijo en su momento, para llevar a cabo esta labor es necesario indicar tres valores críticos: la página de vídeo, la fila y la columna en la que se situará el cursor, de manera que la macro diseñada debe aceptar como parámetros estos valores. En la figura 1 se muestra la forma de resolver el problema planteado mediante la macro *PonCursor*. *Página*, *Fila* y *Columna* son los parámetros necesarios, que serán sustituidos por argumentos adecuados en el momento de invocar la macro. Por ejemplo, la línea *PonCursor 00, 12, 30* pondrá el cursor de la página de vídeo número 0, en la fila 12 y en la columna 30. Como se puede apreciar en este último ejemplo, los argumentos de la macro se escriben en el mismo orden que los correspondientes parámetros y separados por comas.

Los argumentos de una macro pueden ser aquellos operandos ensamblador permitidos en las instrucciones que componen la misma, de forma que es válida, por ejemplo, la línea *PonCursor ch, bl, [si]*; sin embargo, escribir *PonCursor cx, 1000h, WORD PTR [si]* produciría tres errores en el proceso de ensamblado, al ser incompatibles los tamaños de los operandos de la macro con los de esos argumentos.

Las MACROS agilizan la codificación y mantenimiento de un programa

PARÁMETROS POR DEFECTO

En muchas ocasiones puede ocurrir que algún parámetro de los que emplea la macro sea "casi" constante, es decir, que tome un valor concreto con mayor frecuencia que otros. En el caso de la macro anterior, esto es lo que suele ocurrir en la práctica con el parámetro *Página*, que vale 0 la mayoría de las veces.

Cuando se presenta una situación de este tipo, es posible establecer un valor u operando por defecto, de forma que en caso de no escribirse explícitamente un argumento para el parámetro indicado, el programa ensamblador utilice el operando predeterminado al expandir la macro. En caso de indicarse un valor de forma explícita, el operando por defecto será ignorado. Para asociar un valor con un parámetro se utiliza la sintaxis *parámetro:= <valor>*, donde los símbolos '<' y '>' indican al ensamblador que lo comprendido entre ambos debe ser tratado como un literal alfanumérico.

La figura 2 enseña cómo debe ser utilizada esta opción a la hora de declarar el parámetro *Página*. De esta manera, si en el programa se escribe *PonCursor , 12, 30*, al no estar especificado ningún argumento para la página se supondrá que éste es la constante 0. Si se hubiera tecleado *PonCursor 01, 12, 30*, se habría ignorado el valor predeterminado y se habría empleado el 1 como argumento.

Se pueden establecer valores u operandos por defecto para los parámetros de una MACRO

FIJAR LOS PARÁMETROS REQUERIDOS

Ahora bien, ¿qué ocurrirá si no se indica, por ejemplo, un argumento para el parámetro *Fila*? En casos como éste el programa ensamblador asume *que falta un operando* y no sustituye, obviamente, el parámetro implicado por ningún valor. Esto da lugar a mensajes de error en la instrucción o instrucciones afectadas durante la expansión de la macro, pero no en la definición de la misma. Este problema no tiene demasiada importancia, pero puede dificultar un tanto la localización del error cuando la macro es larga y compleja.

Para obligar al ensamblador a generar un error, antes de la expansión de la macro, cuando falta el argumento de algún parámetro, éste se ha de definir como *parámetro:REQ*. La figura 3 muestra cómo queda la línea en la que se declara la macro *PonCursor* al tener en cuenta esto último. En el caso de codificarse la línea *PonCursor 01, , 30*, el ensamblador señalaría un error en la línea que contiene la llamada a la macro, además de en la resultante instrucción incompleta.

MÁS CONTROL: DIRECTIVAS CONDICIONALES

Todavía es posible optimizar aún más el diseño de una macro empleando las directivas de ensamblado condicional, que son las mostradas en la figura 4. Pueden ser empleadas en múltiples circunstancias, pero es aquí donde su utilidad se pone de manifiesto.

Estas directivas, que no deben ser confundidas con las de control de flujo del programa, permiten, mediante condiciones evaluadas a partir del contenido del archivo fuente y/o del modelo de memoria empleado, decidir una forma de ensamblado u otra. En este capítulo se incidirá en aquellas que resulten apropiadas para el desarrollo de las macros que veamos como ejemplos, y se remite al lector a los títulos citados en la bibliografía para su conocimiento en profundidad.

La ejecución de una MACRO es más rápida que la del correspondiente procedimiento

ELIMINAR REDUNDANCIAS

Supóngase que la macro *PonCursor* ha sido extraída de una librería, de forma que el programador conoce qué argumentos deben ser suministrados al invocarla, pero ignora de qué forma trabaja y con qué operandos. En este caso puede ocurrir que, almacenando por algún motivo en los registros BH, DH y/o DL los valores con los que han de trabajar, el programador los emplee como argumentos de la macro escribiendo *PonCursor BH, DH, DL*. Esto es totalmente válido y no causará ningún error, pero el programa ensamblador generará las instrucciones redundantes *MOV BH, BH / MOV DH, DH* y *MOV DL, DL*, que suponen una clara pérdida de espacio y tiempo.

Para eliminar este problema se pueden utilizar las directivas *IFDIF* y *IFDIFI*, cuyo formato es:

```
IFDIF[I] <texto1>, <texto2>
  <sentencias IF>
[ELSE
  <sentencias ELSE>]
ENDIF
```

La primera de ellas da lugar a que se ensamblen las instrucciones del grupo *<sentencias IF>* en el caso de que *texto1* y *texto2* sean diferentes. Esta directiva distingue letras mayúsculas de minúsculas, de manera que al comparar <DH> y <dh> los considera distintos. Por el contrario, *IFDIFI* no tiene en cuenta si los textos están en mayúsculas o minúsculas, y acepta como iguales los dos literales antes mostrados.

Con su uso, la macro *PonCursor* queda tal y como se ve en la figura 5, de forma que, en caso de invocarse la macro mediante la línea *PonCursor BH, DH, DL*, la expansión de la misma se reduciría a las instrucciones *MOV AH, 02h / INT 10h*. Al haberse escogido la directiva *IFDIFI*, da igual escribir con mayúsculas que con minúsculas. Obsérvese que se han empleado los símbolos '<' y '>' para encerrar los textos que se comparan, y no las comillas. Evidentemente, se ha de entender que el valor del parámetro es sustituido por el ensamblador antes de llevar a cabo la comparación, por lo que no se compara, por ejemplo, el texto "Página" con "bh", sino el literal empleado como argumento de *Página*.

LA DIRECTIVAS IFNB, IFB y .ERRx

Con ayuda de las directivas de ensamblado condicional es posible efectuar gran cantidad de tareas. En concreto, mediante las directivas *IFNB*, *IFB* y *.ERRx* se puede abordar el problema de los parámetros por defecto y los parámetros requeridos de una forma bastante elástica y completa.

```
IF[N]B literal
  <sentencias IF>
[ELSE
  <sentencias ELSE>]
ENDIF
```

Si el parámetro encerrado entre '<' y '>' es un conjunto de blancos, la condición comprobada será cierta en el caso de *IFB* y falsa en el caso de *IFNB*. La macro *PonCursor* puede ser modificada, como se ve en la figura 6, para asignar valores por defecto al número de página con ayuda de estas directivas. En el ejemplo se emplea *IFNB*; evidentemente, si se usa IFB es necesario invertir el orden de las acciones.

Aunque en principio el efecto obtenido es el mismo que estableciendo un

FIGURA 4

```
IF
IFE
IFB/IFNB
IFDEF/IFNDEF
IFDIF/IFDIFI
IFIDN/IFIDNI
ELSE
ENDIF
```

FIGURA 5

```
PonCursor MACRO Pagina:=<00h>, Fila:REQ,
Columna:REQ
  mov ah, 02h
  IFDIFI <Pagina>, <bh>
      mov bh, Pagina
  ENDIF
  IFDIFI <Fila>, <dh>
      mov dh, Fila
  ENDIF
  IFDIFI <Columna>, <dl>
      mov dl, Columna
  ENDIF
  int 10h
ENDM
```

FIGURA 6

```
PonCursor MACRO Pagina, Fila:REQ,
Columna:REQ
  mov ah, 02h
  IFNB <Pagina>
    IFDIFI <Pagina>, <bh>
        mov bh, Pagina
    ENDIF
  ELSE
    mov bh, 00h
  ENDIF
  .....
ENDM
```

valor por defecto, las posibilidades de este método son mayores, debido a la gran libertad proporcionada por el empleo de las directivas condicionales. Para demostrarlo nos ocuparemos ahora de los parámetros cuya presencia es obligada introduciendo, además, las directivas de *error*.

El código de una MACRO se duplica con cada llamada a la misma

GENERACIÓN DE ERRORES

De éstas, se emplearán las siguientes:

.ERR [<Mensaje>]
.ERR[N]B <literal1>, [<Mensaje>]

Estas directivas tienen como efecto la generación de un mensaje de error, durante el proceso de ensamblado, de forma incondicional con *.ERR* y condicionalmente con *.ERRB* y *.ERRNB*. Estas últimas fuerzan al programa ensamblador a mostrar el texto indicado en caso de que el valor del parámetro encerrado entre '<' y '>' sea o no blancos, respectivamente. Por el contrario, *.ERR* genera siempre el error, por lo que su uso sólo tiene sentido dentro de una directiva condicional. El empleo de *.ERR*, *.ERRB*, *.ERRNB* y otras similares hace más fáciles la codificación y el mantenimiento de un programa, ya que el implementador puede dar toda la información que considere oportuna sobre los posibles errores. En las figuras 7 y 7 bis se muestra cómo se puede obligar al ensamblador a que considere el parámetro *Fila* como requerido, sin necesidad de emplear la palabra reservada *REQ*. Se puede objetar que este procedimiento es más laborioso, pero la claridad obtenida y la información generada en caso de error compensan el esfuerzo de escribir todas esas líneas una sola vez. Téngase presente, además, que las directivas de error no recargan el archivo ejecutable, debido a que sólo tienen efecto durante el proceso de ensamblado.

FIGURA 7-7bis

```
IFB <Fila>
  .ERR <Se debe depositar en DH la nueva !fila del cursor>
ELSE
  .....
ENDIF

.ERRB <Fila>, <Se debe depositar en DH la nueva !fila del cursor>
```

OPERADORES DE MACRO

Si se observa detenidamente el mensaje de error contenido en las figuras 7 y 7 bis, se puede apreciar cómo aparece el símbolo '!' antes de la palabra 'fila'. El objeto de esto es hacer que el ensamblador tome dicha palabra como un literal alfanumérico y no como el valor actual del parámetro Fila de la macro. De esta manera, si se da el caso de que se deba visualizar el mensaje de error, la palabra 'fila' aparecerá en su posición asignada, mientras que en caso contrario se visualizará un espacio en blanco.

FIGURA 8

Operador	Función
< >	Delimitador de texto
!	Operador literal
%	Operador de expansión
&	Operador de sustitución

En esta situación el símbolo '!', que en otras circunstancias es el operador de negación, funciona como un *operador de macro*. En total hay cuatro de este tipo, los contenidos en la figura 8, y ya se han empleado dos de ellos. Veamos rápidamente cuál es la utilidad de los dos restantes, los operadores de expansión y sustitución.

EXPANSIÓN Y SUSTITUCIÓN

La función del operador de expansión se puede resumir como la de obligar al programa ensamblador a evaluar la representación en formato texto del valor de una expresión, de forma que a partir de ese punto se pueda seguir con la expansión de la macro implicada. El operador de sustitución es necesario cuando se desea sustituir un parámetro por su valor en algunas circunstancias críticas en las que el nombre del parámetro es tomado literalmente, como por ejemplo cuando se encuentra entre comillas, formando parte de una cadena de caracteres. Estos operadores, cómo no, pueden llegar a ser de gran utilidad, pero dado el carácter introductorio de este capítulo no se incidirá en ellos. El lector interesado puede obtener información sobre ellos a partir de la bibliografía señalada.

DIRECTIVAS REPETITIVAS

Hasta el momento hemos visto cómo definir macros para facilitar la tarea de escribir código para nuestro programa. Sin embargo las macros facilitan cualquier tarea en la que aparezca por medio una actividad repetitiva del programador, tanto a la hora de escribir instrucciones como definir variables.

FIGURA 9

```
cubos LABEL BYTE
contador=1
WHILE contador*contador*contador LE 65535
    WORD contador*contador*contador
    contador=contador+1
ENDM
```

Para llevar a cabo esta labor el lenguaje ensamblador dispone de varias directivas:

REPEAT constante
  <sentencias>
ENDM

WHILE expresion
  <sentencias>
ENDM

DIRECTIVA REPEAT

Es la más sencilla de las pertenecientes a este grupo. Su función consiste en forzar la repetición de las sentencias

que contiene el número de veces indicado por la constante, donde ésta puede ser perfectamente el resultado de una expresión numérica.

```
cuadros LABEL BYTE
contador=1
REPEAT 100
  WORD contador*contador
  contador=contador+1
ENDM
```

Como se puede apreciar, después de definir la etiqueta 'cuadrados', que va a servir de punto de referencia para después acceder a los datos, se declara 'contador', que es una *variable en tiempo de ensamblado*. Esto significa que el programa ensamblador trabajará con su valor durante la construcción del fichero objeto, dejando de existir cuando este proceso se dé por finalizado. En el caso del ejemplo anterior, su valor se emplea para inicializar las posiciones de memoria reservadas con la directiva *WORD*. Dado que la constante que acompaña a *REPEAT* es 100, el programa ensamblador actuará como si se hubieran escrito cien líneas *WORD contador*, teniendo en cuenta, además, que el valor de 'contador', que se modifica en cada "ejecución" del bucle se emplea para inicializar cada posición con la cantidad adecuada. Tras el ensamblado del programa se tiene, a partir del desplazamiento indicado por la etiqueta 'cuadrados', una serie de 200 posiciones de memoria que almacenan los cuadrados de los 100 primeros números naturales.

FIGURA 10

Operador	Condición evaluada
EQ	¿Exp1 = Exp2?
GE	¿Exp1 >= Exp2?
GT	¿Exp1 > Exp2?
LE	¿Exp1 <= Exp2?
LT	¿Exp1 < Exp2?
NE	¿Exp1 != Exp2?

Con las directivas .ERRx el programador puede definir sus propios mensajes de error

DIRECTIVA WHILE

Se emplea en aquellas situaciones en las que, por elección o forzosamente, el número de veces que se ha de repetir el contenido de la directiva no está determinado de antemano; por lo demás sus prestaciones son idénticas a las de la directiva *REPEAT*. En la figura 9 se ilustra su manejo mediante el desarrollo de una macro encargada de definir una serie de posiciones de memoria con aquellos cubos de números naturales almacenables en variables de 2 bytes.

Adviértase que la expresión que acompaña a *WHILE* se ha construido con el operador *LE*, nuevo hasta el momento. En la figura 10 se muestran los distintos operadores relacionales que se pueden emplear no sólo en estas circunstancias, sino en cualquier directiva de ensamblado condicional. Si la relación por la que preguntan es cierta, la expresión que se evalúa toma, como conjunto, el valor -1 (cierto). Si esa relación no se cumple el valor de la expresión es 0 (falso).

PARA FINALIZAR

Con lo expuesto hasta el momento se da por terminada esta introducción a las macros. En este capítulo se ha pretendido mostrar al lector las posibilidades de este recurso del lenguaje ensamblador tanto a la hora de escribir código como a la de definir e inicializar variables, y se han apuntado las relaciones existentes entre las macros, sus operadores y las directivas condicionales.

Evidentemente, muchos de estos operadores y directivas se han quedado "en el tintero", ya que lo importante al principio es tener en mente de forma clara la esencia de lo que se puede llegar a hacer con las macros. A partir de este punto, y como en cualquier materia de estudio, el resto se "reduce" a consultar temas específicos.

Las directivas condicionales flexibilizan la codificación de una MACRO

BIBLIOGRAFÍA

-Microsoft Macro Assembler Programmer's Guide-, V 6.0 y sucesivas, Microsoft

Corporation

-Microsoft Macro Assembler Reference. Microsoft Corporation

-Michael Tischer: PC interno 2.0. Programación del sistema. DATA BECKER, 1994

Angulo-Funke:386 y 486-Microprocesadores avanzados de 32 bits. EDITORIAL

PARANINFO

-Scanlon, Leo J.::80286, Programación ensamblador en entorno MS DOS-.

EDICIONES ANAYA MULTIMEDIA, Madrid, 1988.

-Harley Hahn, -Assembler Inside & Out-, Osborne McGraw-Hill 1992

-Miguel Angel Rodríguez-Roselló, - 8088-8086/8087 Programación Ensamblador

en entorno MS-DOS-, EDICIONES ANAYA MULTIMEDIA, Madrid, 1988

-Jürgen Hegner, -8086/8088 Programación en ensamblador-, Ed. DATANET,

Barcelona

LA ESTRUCTURA DE DATOS ARRAY (I)

José C. Remiro

La información se presenta en la realidad con una estructura compleja. Los tipos de datos básicos, aunque útiles, no son adecuados cuando hay que organizar y tratar este tipo de información. Los lenguajes de programación, afortunadamente, presentan estructuras de datos fáciles de utilizar y adaptables a la mayoría de las situaciones. La elección de una buena estructura de datos junto con un cuidadoso diseño de las operaciones a aplicar sobre éstas facilitan la programación.

Hasta ahora solamente se han tratado variables de tipos básicos definidos por el lenguaje de programación (tipos real, entero, carácter y lógico).

Existen situaciones en las que la organización de datos es más compleja, y por tanto, son necesarias estructuras que permitan representar la información a tratar. Surge así el concepto de tipos de datos estructurados, como un conjunto de datos organizado al que están asociados una serie de operaciones básicas que le pueden ser aplicadas.

Podemos clasificar los tipos de datos simples en estáticos y dinámicos, dependiendo de si el tamaño de un tipo de datos se fija en tiempo de compilación, es decir, el tamaño del tipo de datos una vez declarado se mantiene fijo a lo largo de la ejecución del programa, o bien si el tamaño del tipo de datos se fija en tiempo de ejecución, es decir, el número de elementos que forma la estructura, puede crecer o disminuir a lo largo de la ejecución del programa.

Es muy común, encontrar situaciones en la vida cotidiana, en la que los datos están organizados en forma de tabla (un horario, una lista de elementos etc.). Una tabla estaría formada por un conjunto de elementos que tienen relación entre sí y son del mismo tipo (una lista de objetos a comprar, un horario semanal con las tareas a realizar etc.). Este tipo de información es representable en los lenguajes de programación gracias a la estructura *array* (también conocida como matriz, formación, tabla o vector).

DEFINICIÓN DE UN ARRAY

Un *array* es una estructura de datos homogéneos, formada por componentes, todos ellos del mismo tipo (a partir de ahora a este tipo se le denominará tipo base). Además, cada uno de los componentes de la estructura son igualmente accesibles, y pueden seleccionarse de forma directa, indicando la posición que ocupa la componente dentro de la estructura. De esta forma, el tipo de datos *array* tiene asociado un tipo índice, que permite seleccionar todos y cada uno de los elementos que componen la estructura. En la figura 1, se muestra un esquema con la idea intuitiva de *array*, al conjunto global se le ha denominado *HORAS_MES*, contiene 31 componentes (número máximo de días de un mes), cada uno de los componentes del *array* (casilla 1, casilla 2...) representa para cada día del mes un número de horas trabajadas. Siguiendo la terminología utilizada hasta ahora, el número de horas trabajadas pertenecería al tipo base del *array* (en este caso podría ser *real*, si el número de horas trabajadas se representa con decimales), mientras que el tipo índice, se correspondería con un número *entero* entre 1 y 31.

Hay que señalar que el tipo *array* comprende toda la estructura, mientras que cada uno de los elementos pertenece al tipo *real* (en este caso el tipo base).

Se puede seleccionar cada uno de los elementos del *array* , indicando la posición que ocupa dentro del mismo, siguiendo con el ejemplo de la figura 1, para designar el número de horas trabajadas el día 6 del mes representado, basta con que al nombre del *array* le

siga la posición que ocupa dentro del *array* entre paréntesis, es decir, HORAS_MES(6). Es importante, comprender que el valor que representa el acceso a una de las componentes del *array*, pertenece al tipo base del *array*, en este caso la componente HORAS_MES(6) será de tipo *real*, pudiéndose utilizar dicha referencia (o de cualquier otro componente) en cualquier ambiente en que lo pueda ser una variable de tipo real (se le podría asignar un valor de tipo *real*, podría aparecer en una expresión de tipo *real* etc.).

Un aspecto importante del tipo índice asociado al *array* es que indica un orden dentro de la estructura, así podemos acceder a la primera componente, a la segunda componente y así sucesivamente. Esto implica que el índice es calculable, facilitando el tratamiento de las componentes del *array*.

Ampliaremos el pseudocódigo utilizado hasta ahora para permitir la declaración de *arrays*. Para declarar una variable de tipo *array* indicaremos su dimensión (número fijo de componentes que posee el *array*) y el tipo base del *array*, por ejemplo:

TABLA: array (5) lógico;
MES: array (31) real;

La variable TABLA es un *array* de 5 componentes, comenzando a enumerarse por la posición 1 y finalizando en la posición 5, su tipo base es *lógico* (es decir, sus componentes son de tipo *lógico*). La variable MES, es un *array* de 31 componentes y tipo base real.

La notación adoptada para la declaración de variables de tipo *array* es muy genérica, no correspondiéndose con los métodos para definir este tipo de estructuras en los lenguajes de programación.

SELECCIÓN DE COMPONENTES

La declaración de una variable de tipo *array*, crea un nuevo elemento de tipo compuesto. El interés de la declaración de un *array* estriba tanto en manipularlo como una variable que comprende un grupo de elementos del mismo tipo, como en la posibilidad de acceder a cada uno de los componentes del mismo.

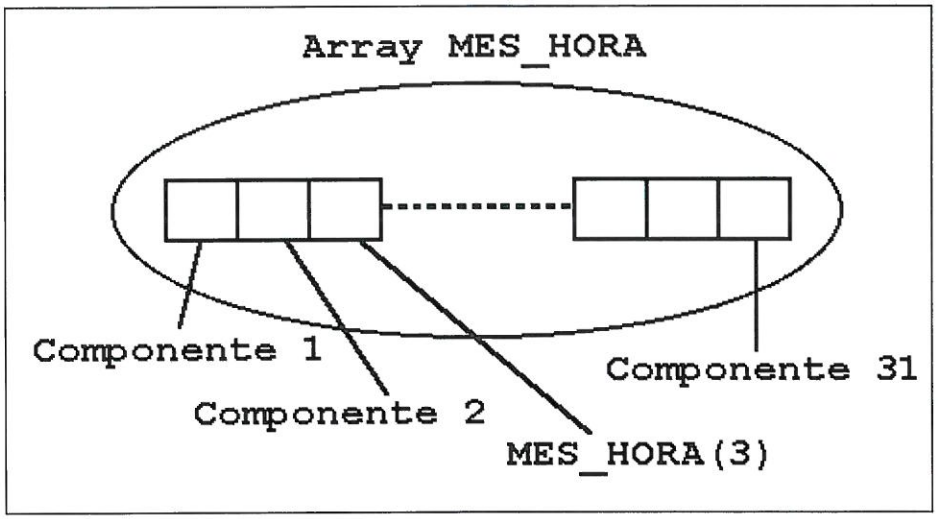


Fig. 1 - Esquema de la estructura array.

En la figura 2, se muestra el pseudocódigo para declarar un variable de tipo *array MES_HORA*, con componentes de tipo *real*, utilizando una variable entera (*INDICE*) para recorrer el *array*. Este algoritmo indica la forma de asignar las horas trabajadas en cada uno de los días del mes y hallar su media.

Con la primera instrucción *desde*, hacemos variar el valor de la variable *INDICE* desde 1 hasta 31 (número de componentes del *array*), introduciendo uno a uno los valores de las componentes (instrucción *leer (MES_HORA(INDICE))*), gracias a la utilización de la variable *INDICE*, la asignación de valores a cada uno de las componentes del *array* se ha simplificado (pensemos que si el índice no fuese calculable, se tendría que haber designado explícitamente cada una de las 31 componentes, es decir, debería haber aparecido 31 instrucciones de asignación, una por componente). Después, con la última instrucción *desde* se acumula en la variable *TOTAL* cada una de las componentes del *array*, finalizando con el cálculo de la media.

Sería posible la utilización de cualquier valor entero entre 1 y 31, o cualquier expresión de tipo entero cuyo valor esté entre los anteriores valores para indicar la componente que se desea manipular. Cualquier valor de índice que no se encuentre entre 1 y el número de componentes declaradas en el *array*, produciría un error en tiempo de ejecución, por lo que hay que tener cuidado a la hora de utilizar índices.

La figura 2 también muestra un algoritmo para calcular el número de horas trabajadas en domingo pidiendo al usuario, al inicio, que introduzca el número de día del primer domingo del mes, accediendo a las componentes del *array* que están a distancia 7, 14... del primer domingo del mes.

UN EJEMPLO

La utilización de estructuras de un lenguaje de programación para construir nuevas estructuras más complejas es una práctica habitual dentro de la programación. Mediante la estructura *array* y construyendo funciones y procedimientos adecuados, se pueden construir estructuras no implementadas por el lenguaje de programación, pero que al programador le pueden ser útiles.

En la figura 3, se muestra la implementación de la estructura conjunto de caracteres alfabéticos. Se utilizan *arrays* de 26 elementos (número total de caracteres alfabéticos eliminando la 'ñ') para representar un conjunto. Se han definido los procedimientos y funciones para implementar las operaciones más usuales entre conjuntos (unión, intersección, pertenencia etc.).

Se puede observar las operaciones básicas que se pueden aplicar sobre los *arrays* (selección de elementos, inserción de un elemento en un *array*, búsqueda de un elemento dentro del *array* etc.), dejaremos para un artícu-

```
MES_HORA: array (31) real
media, total: real
indice: entero
inicio
 Desde indice = 1 hasta 31
     leer(MES_HORA(indice))
 total = 0
 Desde indice = 1 hasta 31
         total = total + MES_HORA (indice)
 media = total / 31
 Escribir (' La media de horas trabajadas es ',
media)
fin

procedimiento total_domingos (mes: array(31)
real; var total: real)
  i: entero
  primer_domingo: entero
inicio
 total = 0
 Repetir
    Escribir('Introducir el primer domingo del mes
')
     leer (primer_domingo)
 hasta (primer_domingo > 0) y (primer_domingo
< 8)
 Mientras primer_domingo <= 31
     total = mes (primer_domingo)
     primer_domingo = primer_domingo + 7
 Fin Mientras
Fin
```

Fig. 2 - Algoritmo de ejemplo.

lo posterior el tema de la ordenación de los elementos de un *array*.

Para la representación de un conjunto se utiliza un *array* que contendrá los elementos que forman el conjunto situados en las primeras posiciones, por ejemplo, si el conjunto fuese {R, B, A}, el *array* contendría en sus tres primeras componentes a las letras correspondientes, el resto de las componentes del *array* contendrían el carácter " " (carácter blanco o espacio).

El procedimiento *INICIA_CONJUNTO*, toma como argumento un *array*, asignando a cada componente el carácter blanco, el resultado obtenido es el conjunto vacío (todos los elementos del *array* tienen el carácter blanco).

La función *CARDINAL*, dado un *array*, cuenta el número de componentes hasta que encuentra una componente que contiene el carácter blanco, es decir, a efectos de operación sobre conjuntos calcula el número de elementos del conjunto. Es importante, haber definido previamente la estructura de la representación de conjunto.

La función *PERTENECE*, dada una variable de tipo carácter, comprueba si el contenido de ésta se encuentra en alguna de las posiciones "significativas" del *array*, es decir, a lo largo de las componentes del *array* cuya posición no supera al cardinal del conjunto. De nuevo se señala la importancia de comprender que todas y cada una de las componentes de un *array* pertenecen al tipo base (en nuestro caso tipo carácter).

La función *INCLUIDO* comprueba si, dados dos *arrays*, el primero de ellos está incluido en el segundo. Utilizando la función *PERTENECE*, comprueba si para cada elemento del primer conjunto pertenece al conjunto representado por el segundo *array*. Fijémonos, como una planificación previa de los procedimientos, nos permite aprovecharlos posteriormente (tema de algún que otro artículo anterior).

El procedimiento *UNION* almacena en un tercer *array* los elementos no duplicados de los dos primeros *arrays*. Si hubiese dos elementos iguales en el primer y segundo conjunto, entonces en la unión de éstos solamente aparecería una vez (la implementación es consistente con la idea matemática de conjunto). No se hará ningún comentario acerca del procedimiento *INTERSECCIÓN*, por tener similar estructura.

Por último, el procedimiento *INTRO_DATOS_CONJ* se encarga de aceptar desde el teclado las letras mayúsculas que formarán el conjunto introducido como parámetro. Habrá de comprobar que el usuario no introduce más que letras mayúsculas y que efectivamente no se introducen elementos duplicados. La función *CÓDIGO_ASCII* supondremos que, dado un carácter, devuelve el código ASCII en decimal correspondiente (las letras mayúsculas se encuentran en el código ASCII entre los números 65 y 90, que coinciden con la condición de la instrucción *si* más externa). En Turbo Pascal la función que realiza este proceso es *Ord*, en C la comprobación es directa y en Qbasic es *asc*.

LENGUAJES Y ARRAYS

A continuación realizaremos algunos breves comentarios acerca del empleo de los *arrays* en los lenguajes de programación C, Turbo Pascal y Qbasic, utilizando para ello los ejemplos dados en pseudocódigo.

Comenzaremos por Turbo Pascal que permite utilizar de una manera muy elegante los *arrays*, gracias a que es un lenguaje fuertemente "tipado". Para representar estructuras mediante *arrays* podemos utilizar la sección de definición de tipos. En el ejemplo de la figura 3. se podría haber definido:

```
. . . . .
const
 max_cardinal = 26;
type
 indice = 1..max_cardinal;
 elemento = 'A'..'Z';
 conjunto = array[indice] of elemento;
var
 ejem_conj: conjunto;
 cadena: array [1..12] of char;
. . . .
```

Las consecuencias de esta declaración es la ocultación de la definición del nuevo tipo de datos *conjunto* y la verificación de asignaciones a las variables que se definan posteriormente con los tipos definidos por el programador. Por ejemplo, si se hubiese intentado asignar un valor 0 a una variable de tipo *índice* en tiempo de compilación, se habría producido

Un *Array* es una estructura de datos homogéneos, formada por componentes, todos ellos del mismo tipo

```
procedimiento inicia_conjunto (var conjunto:
array(27) caracter)
 indice: entero ;
inicio
  Desde indice = 1 hasta 27
          conjunto(indice) = ' '
fin

funcion cardinal (conjunto: array(27) caracter):
entero
  indice : entero ;
inicio
 Si conjunto(1) = ' '
     cardinal = 0
   en otro caso
     indice = 1
     mientras (conjunto(indice) <> ' ') y (indice <=
27)
              indice = indice + 1
     indice = indice - 1
 Fin Si;
fin;

función pertenece (elem: caracter; conjunto:
array (27) caracter): logico
 num_elem, indice: entero;
 b: logico;
inicio
  num__elem = cardinal(conjunto)
  b = falso
  indice = 1
  Mientras (num_elem <  indice ) y ( no b)
       Si conjunto (indice) = elem
          b = vardadero
        en otro caso
          indice = indice + 1
  Fin mientras
  pertenece = b
 fin

funcion incluido (conj_A, conj_B: array(27)
caracter): logico
  num_elem_A, indice: entero
  b: logico
inicio
  num_elem_A = cardinal (conj_A)
  b = verdadero
  indice = 1
  Mientras (indice <= num_elem_A) y b
     b = b y pertenece (eonj_A(indice), conj_B)
     indice =  indice + 1
  Fin Mientras
  incluido = b
fin

procedimiento union (conj_A, conj_B: array
(27) caracter;
                          var conj_UNI: array
(27) caracter);
 indice : entero
cardinal_A, cardinal_UNI: entero
inicio
  conj_UNI = conj_A
  cardinal_A = cardinal (conj_A)
  cardinal_UNI =  cardinal (conj_UNI)
  Desde indice = 1 hasta cardinal(conj_B)
       Si no ( pertenece (conj_B (indice),
conj_UNI ))
            cardinal_UNI = cardinal_UNI + 1
            conj_UNI (cardinal_UNI) = conj_B
(indice)
       Fin Si
   Fin Desde
fin
```

```
procedure intersección (conj_A, conj_B: array
(27) caracter; var conj_INTER: array (27)
caracter)
 indice: entero
 cardinal_INTER: entero
inicio
  inicia_conjunto (conj_INTER)
  cardinal_INTER = 0
  cardinal_A = cardinal (conj_A)
   Desde indice = 1 hasta cardinal_A
          Si pertenece (conj_A(indice), conj_B)
              cardinal_INTER = cardinal_INTER +
1
              conj_INTER(cardinal_INTER) =
conj_A(indice)
          Fin Si
   Fin Desde
Fin

procedure intro_datos_conj (var conjunto:
array(27) caracter)
 indice: entero
 aux: caracter
inicio
 inicia_conjunto (conjunto)
 indice = 0
 Repetir
   escribir('Introduzca un  carácter (fin = barra
espaciadora)  ');
    leer(aux)
    Si (codigo_ASCII (aux) <= 90 ) y (
codigo_ASCII(aux) >= 65)
        Si no (pertenece (aux, conjunto))
            indice = indice + 1
            conjunto (indice) = aux
       Fin Si
    Fin Si
  Hasta aux = ' '
fin
```

Fig. 3 - Algoritmos de la estructura conjunto, implementada con arrays.

un error. Si en tiempo de ejecución hubiésemos intentado asignar a una componente de una variable de tipo conjunto un carácter no alfabético en mayúsculas, se hubiese producido otro error. Además, el mantenimiento del programa, si se hubiese decidido aumentar los elementos posibles de un conjunto a las letras minúsculas, hubiese sido mínimo con la modificación de *max_cardinal* (ponerlo con valor 52), y aumentando el tipo *elemento*. De esta manera, el programa podría haber seguido funcionando. Estas características se deben tener en cuenta, si el lenguaje en el que estamos aprendiendo a programar es Pascal. Directamente se pueden definir variables de tipo *array* (ver anterior definición). Si se va a implementar una estructura de datos utilizando *arrays*, merece la pena crearla mediante la declaración de tipos facilitada por Pascal.

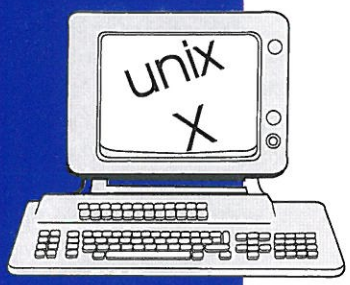
Si el lenguaje seleccionado es C, nos encontramos con un método de definición muy sencillo. Lo primero que debe indicarse es el tipo de base, seguido del nombre del *array* y al número de elementos de éste encerrado entre corchetes. Debe tenerse en cuenta que en C los *arrays* comienzan en la posición ø. Por ejemplo, si declaramos *int conjunto[26]* , los elementos se referenciarán con posiciones 0 a 25. En este lenguaje, los punteros están muy relacionados con el tratamiento de *arrays*. Un punteros es una variable que contiene una dirección. Así, si a continuación de la anterior declaración, declaramos *int *ptr*, se definirá un puntero. La asignación *ptr =&a[0]*, hace que el puntero *ptr*, apunte al elemento 0 del *array*, y la instrucción *x=*ptr*. copiará el contenido de *a[0]*en *x*. Con incrementos sucesivos de *ptr*, se iría apuntando a los siguientes elementos del *array*.

En Qbasic se utiliza la sentencia *DIM* para declarar los *arrays*. Éstos pueden ser estáticos o dinámicos. En este último caso, el número de componentes puede crecer en tiempo de ejecución. Esta situación, no es posible ni en C, ni en Pascal los *arrays* además, no puede tener un subíndice negativo (cosa que sí puede suceder en Pascal, pero no en C) ni estar fuera de rango. El rango puede comenzar por cualquier número positivo. Hacemos referencia a las variables, mediante el nombre del *array* y el valor del índice para la componente a acceder. Además, contiene una serie de funciones especiales para la utilización de *arrays* (*LBOUND, EBOUND*, etc.).

EL PRÓXIMO ARTÍCULO

Tras este panorama general sobre la estructura de datos *array*, en la próxima entrega completaremos este tema con *arrays* de varias dimensiones y *strings* (cadenas de caracteres).

Sería un buen ejercicio implementar, en el lenguaje que se halla elegido para aprender a programar, los ejemplos de la figura 3 creando nuevos procedimientos y funciones sobre conjuntos.



LAS SHELLS (I) INTÉRPRETES INTERACTIVOS DE COMANDOS

Fernando J. Echevarrieta

Un sistema operativo es mucho más que un conjunto de comandos inconexos. De hecho una *shell*, además de intérprete del usuario, dota de estructura propia a todo este conjunto de órdenes y, aún más allá, como interfaz con el núcleo, puede servir como puerta de entrada a la presentación de ciertos conceptos de la arquitectura interna del sistema sobre el que se apoya. Mediante soluciones a algunos problemas tan simples como realizar algunas consideraciones sobre el *path* o estudiando como se ejecuta un comando, en el presente artículo se introducirán otros temas, como la estructura del fichero de *passwords*, posibilidades de modificación de la información que el sistema tiene sobre un usuario, protección de programas en *background* e ideas más avanzadas sobre los mecanismos internos de UNIX, como el sistema de generación de procesos.

Para el usuario la shell tiene una doble función: interfaz con el núcleo y lenguaje de programación

Hasta el momento se han estudiado distintas características aisladas de la superficie del sistema. Con el estudio de las shells, se dotará de estructura a todo lo expuesto y se comenzará a introducir algunos conceptos avanzados a través de la exposición de la interfaz shell-núcleo.

PANORAMA GENERAL DE SHELLS UNIX

Desde el punto de vista del usuario, las *shells* cumplen una doble función: por una parte representan una interfaz con el sistema y, por otra, son en sí lenguajes de programación interpretado. En cualquier caso, se recuerda una vez más que para el sistema operativo, las *shells* no son más que simples programas de usuario. Como ya se adelantó en su día, como cualquier otro programa, ninguna de ellas en concreto es necesaria, por lo que, con ciertas salvedades, cada usuario puede optar por la que desee. La más famosa, por ser la primera, es la *Bourne shell* o, simplemente *shell* o *sh*. Tras ella, surgió la *csh* (*c-shell*) con la versión de Berkeley, que recibe este nombre debido a su sintaxis, similar a la del lenguaje C. A partir de ellas han ido surgiendo otras que incorporaban cada vez más mejoras. Así, la sh tuvo su descendencia en la *ash*, *ksh* (*Korn shell*) y en la actualmente muy de moda *bash* (Bourne again sh). La bash es la *shell* que se utiliza por defecto en Linux. Por su parte, la csh creció con la *tcsh*. En general, cuando se habla de *shell* en UNIX se alude a cualquiera de ellas, pero bastará conocer sh y csh para moverse con cualquier *shell* "clásica" como pez en el agua. Sin embargo, el mundo de las *shells* sigue creciendo y, aunque de facto únicamente se utilicen para la programación, lenguajes interpretados más modernos como *perl* o *tcl* (con la herramienta *tk* para X WINDOW), también forman parte de este especial grupo de programas.

LA SHELL COMO AUTÓMATA

En el modo de operación normal, la *shell* es el programa que aparece inmediatamente después del *login*, una

vez se entra al sistema. El programa de *login* "sabe" que *shell* debe lanzar a cada usuario ya que esto aparece indicado junto al resto de los datos del mismo en su entrada en el fichero */etc/passwd*.
Un ejemplo de línea tipo en este fichero sería:

echeva:wpDWOjV1skjtU:4033:14:Fernando J. Echevarrieta Fernandez:/usr/sanson/echeva:/usr/local/tcsh

donde los campos, separados por dos puntos (:) son, respectivamente: *login* (identificador de usuario), *password* encriptado, *uid* (número IDentificador de Usuario), *gid* (número IDentificador de Grupo principal), información de usuario, directorio *home* y *shell* de *login*.

Aunque uno no sea el administrador del sistema, dispone de la posibilidad de cambiar su *shell* de *login* mediante el comando *chsh*. En principio, cualquier programa podría servir como *shell* ya que el sistema operativo no hace distinción. De hecho, esto es lo que se hace en la mayoría de las BBS accesibles mediante el *telnet* de Internet. Sin embargo y por razones de seguridad, únicamente se podrá elegir como *shell* aquellos programas que figuren en el fichero */etc/shells* que sólo puede modificar *root*. Del mismo modo, si el administrador lo permite, podrá cambiarse la información de usuario mediante el comando *chfn*.

La *shell* correspondiente es invocada desde el proceso de *login* mediante *exec*, una llamada al sistema fundamental en UNIX que se tratará en el artículo dedicado a procesos. Una vez en ejecución, realizará unas acciones u otras en función de que *shell* se trate. Así pues, la familia sh ejecutará el fichero */etc/profile* y después buscará en el directorio *home* el fichero *.profile* para hacer lo propio. En el primero es donde *root* suele configurar unas opciones estables y uniformes para todos los usuarios y el segundo es el que cada uno puede utilizar para personalizar su propia *shell*. Si por el contrario, la *shell* es de la familia csh, ejecutará secuencialmente los ficheros *.login* y *.cshrc* del directorio *home* correspondiente (también pueden existir ficheros globales como /etc/csh.cshrc). Así pues, se puede decir que /etc/profile y ~/.profile (~ para la *shell* designa al directorio *home*) son a sh (y familia), como ~/.login y ~/.cshrc a csh (y familia) y como el AUTOEXEC.BAT al COMMAND.COM, la *shell* del DOS.

Aunque un usuario no sea root podrá cambiar su shell y la información que de él tiene el sistema

Llegados a este punto, es importante realizar una primera clasificación entre *shells* de *login* o *login shells* y *shells* ordinarias. Una *shell* de *login*, como su nombre indica, se genera a partir del proceso de *login*, mientras que una ordinaria se genera por otro motivos: clonación de una *shell* de *login*, invocación como comando por parte del usuario, generación de un xterm, etc. Una *shell* que no es de *login* nunca leerá el fichero .login, por lo que es importante que aquellas partes de la configuración que deban encontrarse en toda *shell* (por ejemplo, el ajuste del *path*) figuren en el .cshrc mientras que otras partes, como los avisos de si existe nuevo *mail*, figuren únicamente en el .login.

Para ejecutar un comando externo la shell se clonará a sí misma y su hija quedará recubierta por el ejecutable

Una vez la *shell* ha arrancado y leído su configuración comenzará un bucle sin fin representado en la figura 1. En ella, se pueden apreciar cuatro estados principales representados por rectángulos: 1. Mostrar Prompt, 2. Leer (Parser), 3. Procesar la Entrada y 4. Inicio de Operaciones. Existe otro estado, que se ha representado con una elipse de línea discontinua, que es el de espera de fin de operaciones. Como el lector podrá adivinar, se debe a que no siempre se pasa por el mismo: en caso de que en la línea de comando se haya introducido un comando terminado en &, éste se ejecutará concurrentemente en *background*, por lo que la *shell* volverá inmediatamente a mostrar el *prompt* sin perder el control del terminal ni tener que esperar a la finalización del programa lanzado.

EL PARSER: ESTRUCTURA DE COMANDOS.

Todos los comandos que se han visto hasta ahora, obedecen a una estructura que es analizada por la *shell* (*parsing*) y responde a la forma *nombre [-opciones] [argumentos]* (obsérvese el orden) donde el nombre del comando casi siempre hace referencia a un fichero, ya sea a través de un *path* absoluto (*/bin/ls*), relativo (*bin/ls*) o se deja sin indicar para su búsqueda en el PATH (*ls*). Las opciones casi siempre irán precedidas de un guión -, y estarán constituidas por una sola letra (ls -a) y casi siempre (UNIX es el S.O. del "casi siempre") serán acumulativas (*ls -al* equivale a *ls -a -l*). En ocasiones, incluso las opciones podrán tener sus propios argumentos. Por ejemplo, en la línea *tar cvfb fich 1024 dir*, *fich* es el argumento de la opción *v* y *1024* (factor de bloque) es el argumento de la opción *b*, mientras que *dir* es el argumento del comando (remítase el lector a *man tar* para mayor información). El ejemplo también sirve para ilustrar que "casi siempre" y "siempre" no es lo mismo ya que en este caso, las opciones no se han precedido de un guión (comando excepcional en el que es opcional). Por último, los argumentos podrán ser desde

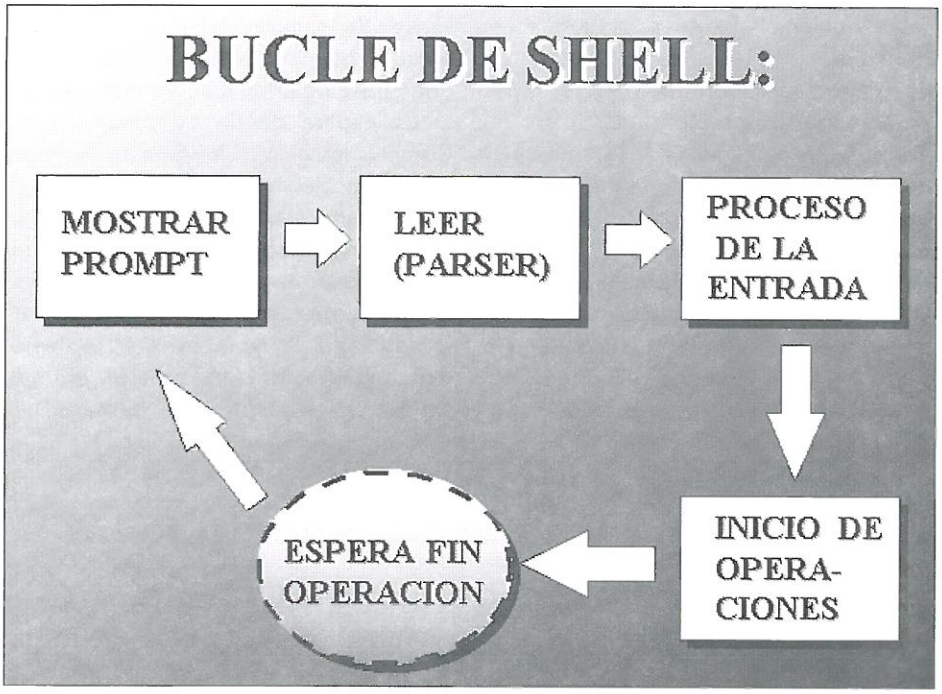


FIGURA 1: Bucle de shell.

ficheros (*ls .newsrc .oldnewsrc*), pasando por usuarios (*talk echeva*) y hasta *hosts* (*rlogin sanson, ftp ames.arc.nasa.gov*).

Una característica de la *shell* de gran utilidad a la hora de realizar programas legibles, es que si una línea se termina con un *backslash* \, la *shell* interpretará que la línea de comando continúa en la siguiente línea.

ENTORNO DE COMANDOS

Del mismo modo que se ha podido comprobar que todos los comandos UNIX responden a una misma estructura, se puede estudiar su entorno. Así las entradas podrán ser sus propias opciones y argumentos, ficheros predefinidos (como *.profile*, *.mailrc*) y la entrada estándar. Pero en esta ocasión se tratará el último mecanismo de entrada que son las variables de entorno como TERM, PAGER o PATH. Análogamente, la salida de un comando podrá ser un fichero (caso de vi o cc), la salida estándar o la de error y, el último mecanismo que será muy útil a la hora de programar en *shell*: el valor de retorno.

INICIO DE OPERACIONES: EJECUCIÓN DE UN COMANDO

Las *shells* suelen distinguir dos tipos de comandos: internos y externos. Mientras los primeros constituyen parte de la propia *shell* manifestándose a través de subrutinas de la misma, los segundos lo hacen a través de programas ejecutables, lo que implica la ejecución de un proceso ajeno a la *shell*. Así pues, al introducir un comando, una *shell* primero comprobará si es interno mirando entre sus propias subrutinas y, en caso de no encontrarlo, considerará que la primera palabra que figuraba en línea de comando es el nombre de un programa ejecutable que buscará en el *path* si no viene precedido de una ruta absoluta o relativa. Una vez encontrado el ejecutable, la *shell* se clonará a si misma y generará una *shell* idéntica hija que será inmediatamente recubierta (sustituida) por el ejecutable [CONCEPTO 1]. Esta secuencia de generación de procesos que puede, en principio, parecer algo extraña, es fundamental en UNIX y se estudiará detenidamente en su momento. Una vez se ha iniciado la ejecución del programa en cuestión pueden darse dos situaciones como se comentó en el bucle de *shell*: a) que la *shell* interactiva (madre) espere la finalización del ejecutable y b) si la línea de comando terminaba con un *ampersand* & (ejecución asíncrona en *background*), que continúe el bucle. En cualquier caso, hay que mencionar que las *shells* UNIX disponen de la característica *type ahead*, que consiste en almacenar en un *buffer* todo lo que se teclea aunque aún no hayan mostrado el *prompt*, por lo que se puede continuar tecleando aunque un comando no haya finalizado su ejecución.

A la hora de lanzar procesos en *background* se pueden citar los comandos *nohup <comando> &* que permite que el comando se siga ejecutando aunque termine la sesión y se desconecte el terminal (en tcsh y bash, no es necesario) y *nice <comando> &* que realiza la ejecución de comando con una menor prioridad.

Existen dos tipos de shells: ordinarias y de login

CONSIDERACIONES SOBRE EL PATH

Como se ha venido teniendo en cuenta de modo más o menos tácito hasta el momento, en todas las *shells* UNIX se define una variable *PATH* (familia sh) o *path* (familia csh), análoga a la de DOS, que determina un camino de búsqueda de ejecutables en la estructura de ficheros. En este punto y dando vueltas a una idea tan simple, se introducirán otras más complejas sobre las interioridades del sistema. Cada *shell* trata de una forma distinta las variables. Así, para definir el valor del *path* en una *shell* sh se indicará:

PATH=dir1:dir2:...:dirn

mientras que en una *shell* csh habrá que indicar:

set path = (dir1 dir2 ... dirn)

A partir de la csh, todas las *shells*, construyen una *tabla hash* con todos los ejecutables que se encuentran en el *path* (NOTA: Una tabla *hash* es una herramienta de programación que consiste en definir una estructura de

datos que se presenta al usuario con un tamaño mucho mayor del que en realidad tiene con el fin de permitirle una colocación mucho más cómoda de los datos y aprovechando que es prácticamente imposible que el usuario trate de llenarla. El caso típico es el de las hojas de cálculo. Para el almacenamiento de los datos la estructura debe ir acompañada de una función que realice un "mapeado" de las posiciones "visibles" a las "reales" y que, por supuesto será de N a 1, por lo que también deberá ser capaz de resolver *colisiones* o casos en los el usuario escribe sobre dos direcciones "visibles" que se corresponden con la misma "real"). Esto reporta la gran ventaja de que una búsqueda en el *path* no implica accesos a disco, la parte más lenta del sistema que, además ahora, es un recurso compartido. Sin embargo, como contrapartida suele provocar verdaderos quebraderos de cabeza a los usuarios noveles que crean nuevos programas, y al tratar de ejecutarlos comprueban que el sistema no es capaz de encontrarlos. En efecto, en numerosas ocasiones el usuario comprueba que no ha dotado de permiso de ejecución al nuevo programa, la contrariedad surge cuando tras reparar este error, el problema persiste. Si esto ocurre, lo más probable es que se deba a que la *shell* busca el nombre del programa en el *path* que se encuentra reflejado en su tabla *hash*, que fue creada antes que el nuevo ejecutable [CONCEPTO 2]. Para solventarlo, habrá que regenerar la tabla mediante *hash -r*, desde una bash o *rehash* desde una csh/tcsh.

Alternativas posibles son: generar una nueva *shell* mediante el comando *exec* (exec bash, por ejemplo) o realizar *login* otra vez (lo que generará una nueva *shell*), para lo que no hará falta salir y volver a entrar ya que se puede ejecutar directamente el programa *login*. El comando exec "mapea" directamente sobre la llamada al sistema *exec* y provoca la sustitución del programa que la invoca (en este caso la *shell*) por el invocado (la nueva *shell*) [CONCEPTO 3]. Una última alternativa es regenerar la tabla mediante la reasignación del valor de la variable *PATH* o *path*, para lo que la forma más cómoda consistirá en ejecutar de nuevo el fichero de configuración de la *shell* que defina este valor (léase .login, .cshrc o .profile) del mismo modo que si en DOS se ejecutara de nuevo el AUTOEXEC.BAT.

Todas las shells permiten redirigir los dos primeros descriptores de fichero

Aquí el problema surge del inicio de operaciones típico de UNIX para la ejecución de un programa que se ha descrito anteriormente (CONCEPTO 1). Si se ejecuta uno de estos ficheros de configuración directamente, se generará una nueva *shell* hija de la actual para interpretarlo, y será esta *shell* la que reciba la nueva configuración. Así pues, una vez haya terminado de interpretar el fichero en cuestión, morirá y retornará con éxito devolviendo el control a la *shell* madre desde la que se inició la operación y cuya configuración habrá permanecido inalterada. Para evitar que para interpretar el fichero de configuración se genere una *shell* hija y, de este modo, todos los cambios afecten a la presente, se podrá utilizar el comando *source* indicando, por ejemplo, *source .cshrc*.

El entorno de un comando define cuatro mecanismos de entrada y tres de salida

Una última consideración sobre el valor del *path* es que, debido a que el "directorio actual" (.) es variable, no se puede encontrar almacenado en la tabla *hash*. Por ello, si se desea incluirlo en el *path*, es conveniente hacerlo en último lugar ya que, en caso contrario, al realizarse las búsquedas en el *path* por orden, todo intento de ejecución de un comando implicaría un acceso a disco para leer el directorio actual antes de utilizar la tabla *hash*, con lo que esta perdería su sentido. De hecho, es aconsejable no incluir este directorio en el *path* ya que se cae en el riesgo de ejecutar un programa que algún avieso personaje haya creado en el directorio actual con el nombre de un comando del sistema. Programa que al ser ejecutado tomaría los permisos de quien lo lanzara con todas la implicaciones que eso conlleva, especialmente si, quien lo ha ejecutado por tener el . en el *path* ha sido el *root*.

TABLA 1

< fichero	Redirige STDIN desde fichero
> fichero	Redirige STDOUT a fichero
>> fichero	Añade STDOUT a fichero
>! fichero	Aunque fichero exista, redirige
>>! fichero	Si fichero no existe, lo crea
<< cadena	Resto -> STDIN hasta encontrar cadena

Mecanismos de redirección independientes de shell.

COMPOSICIÓN DE ORDENES Y CONTROL DE EJECUCIÓN

La *shell* permite agrupar comandos mediante tres mecanismos: El terminador secuencial o punto y coma (;)

comando1 ; comando2

ejecuta el comando2 una vez ha terminado el 1. El terminador paralelo o *ampersand* (&)

comando1 & comando2

ejecuta ambos comandos en paralelo. Y, por último, el *pipe*, estudiado en anteriores ocasiones

comando1 | comando2

Un tipo de agrupamiento especial es el que se realiza por medio de paréntesis

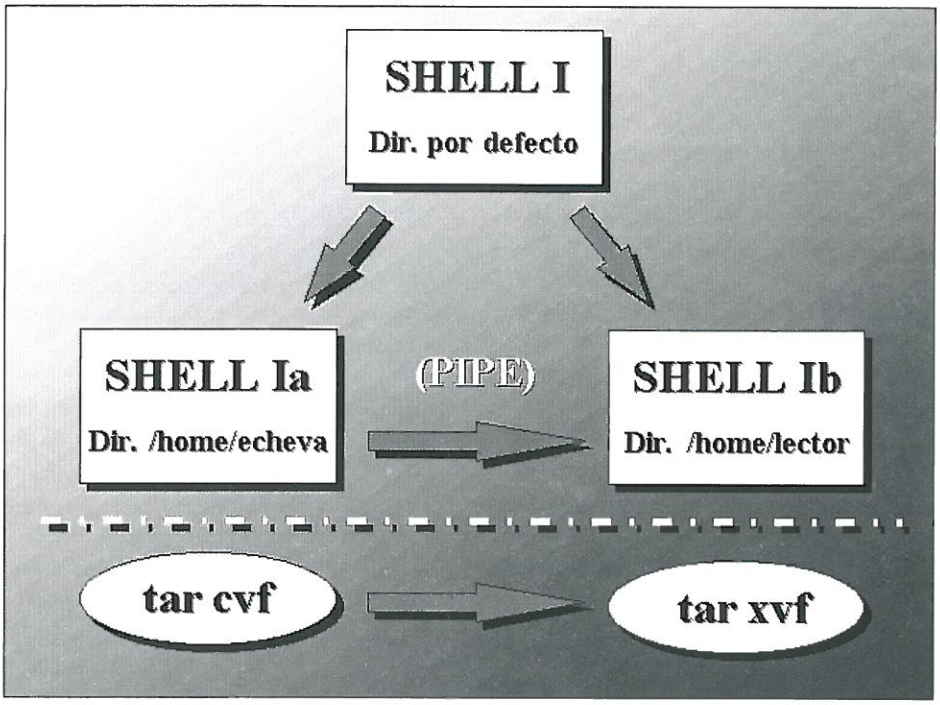


FIGURA 2: Ejecución de dos shells en paralelo conectadas mediante un pipe.

(agrupación de comandos)

La *shell*, al encontrar un paréntesis, genera una *shell* hija clónica de sí misma y es esta *shell* la que actuará interpretando la agrupación de comandos y no la *shell* original. Así pues, si se teclea *(cd /usr/local)*, el efecto será la generación de una *shell* hija que tomará como directorio por defecto el /usr/local y después morirá devolviendo el control a la *shell* madre en la que el usuario continuará sin haberse cambiado de directorio. Estos efectos, en principio, pueden producir rechazo al nuevo usuario, sin embargo, esto se debe a la falta de costumbre ante la potencia de un sistema operativo multiproceso como UNIX, por lo que es fundamental entenderlos para luego obtener todo el provecho. Así, un ejemplo de aprovechamiento de esta característica podría ser:

(cd /home/echeva ; tar cvf - .) | (cd /home/lector ; tar xvf -)

representado en la figura 2. Al analizar esta línea de comando, la *shell* detectará el *pipe*, por lo que se preparará para iniciar dos procesos en paralelo e interconectados. A ambos lados del mismo se encuentran sendos paréntesis, así que lo que se iniciarán serán dos *shells* clónicas hijas de la original de modo que la salida estándar de la primera se encontrará conectada a la entrada estándar de la segunda. Cada *shell* comenzará su propio bucle e interpretará el contenido del paréntesis asociado. Así, a la vez que una entra en el directorio */home/echeva*, la otra lo hará en el */home/lector*. Tras ello, la primera comenzará a convertir toda la jerarquía de ficheros que "cuelga" de ese directorio en uno único que devolverá por la salida estándar (parámetro - para la opción f del *tar*) mientras que la segunda, a la vez, estará ya recogiendo desde el comienzo este flujo que le llega por la entrada estándar y convirtiéndolo de nuevo en una jerarquía de ficheros en "su" directorio. Todo el proceso se realiza concurrentemente por lo que desde el momento en que la primera *shell* produce el primer ítem de información, este ya está siendo recogido por la segunda. Cuando se estudien mecanismos de red se verá que con una simple modificación de esta línea se podrán transferir sistemas enteros de ficheros de un punto a otro del mundo.

Tal es la potencia del sistema y, aunque la *shell* proporciona estos mecanismos, cumpliendo su labor como interfaz del núcleo, se recuerda una vez más que es este último el que realiza toda la gestión de procesos. Así, aunque las *shells* de UNIX se encuentran disponibles también para DOS, se verán limitadas por este sistema y nada de esto será posible.

También es posible realizar un control de ejecución mediante los operadores:

comando1 || comando2

que ejecuta comando2 si y sólo si comando1 falla y

comando1 && comando2

que ejecuta comando2 si y sólo si comando1 acierta.

REDIRECCIÓN E/S INDEPENDIENTE DE SHELL

Como se mencionó en anteriores artículos, cada proceso UNIX dispone de tres *descriptores de fichero* asociados. Un descriptor de fichero es un puntero que señala a una estructura que representa al fichero (recuérdese una vez más, que para UNIX todo son ficheros). El número 0 corresponde a la entrada estándar (STDIN), el 1 a la salida estándar (STDOUT) y el 2 a la salida de error (STDERR). La mayoría de los programas, en lugar de realizar accesos directos a ficheros de dispositivo, utilizan estos descriptores, por lo que la funcionalidad de la *shell* de redirigirlos es sumamente importante. La tabla 1 muestra los mecanismos de redirección independientes de *shell*.

A la hora de redirigir la salida a un fichero puede surgir el problema de la destrucción de los datos que ya contenga ya que se creará de nuevo. Para protegerse de esto, la csh definió una

Para poder ejecutar un nuevo comando del path será necesario regenerar la tabla hash

TABLA 2	
!	Activación sustitución history.
!n	Evento con número de secuencia n.
!-n	n-ésimo evento anterior.
!!	Ultimo evento (lo mismo que !-1).
!prefijo	Más reciente comenzando por prefijo.
^xx^yy	Cambia xx por yy en el último.
!*	Todos los argumentos del último evento.
!$	Último argumento del último evento.
!^	Primer argumento del último evento.
!:n	n-ésimo argumento del último evento.
evento:s/xx/yy/	Cambia xx por yy en el evento indicado.

Sustituciones de history más frecuentes.

BIBLIOGRAFÍA

- "Unix System V"; Antonio P. Gimenez; Anaya Multimedia, 1994

- "Unix"; George W.Leach; Addison Wesley, 1994

-"¡Aprenda Unix Ya!"; Jim Gardner; Anaya Multimedia, 1993.

- "Unix para impacientes"; Abrahams y Larson; Addison Wesley, 1995.

- "El libro del SCO UNix"; Joanne Woodcock y Michael Halvorson; Anaya Multimedia, 1995.

variable de *shell* binaria llamada *noclobber*. Así, si se activa mediante *set noclobber* (en bash *set -o noclobber*), la *shell* no permitirá realizar redirecciones destructivas (>) a ficheros que ya han sido creados. En cualquier caso, como indica la tabla 1, se podrá forzar la redirección mediante el uso del operador "por narices" (!). El efecto protector de la variable, a la hora de añadir la salida a un fichero, se manifiesta prohibiendo la operación si el fichero no ha sido creado aún. De nuevo, la trampa consiste en la aplicación del operador "por narices". Como toda variable binaria, se puede desactivar mediante *unset noclobber*.

Otro mecanismo muy interesante, sobre todo a la hora de realizar programas en *shell*, es la capacidad de redirigir la propia línea de comando (o el texto del propio programa) hacia la entrada estándar. Para ello se utiliza el operador << *cadena*. Cuando la *shell* encuentra este operador, deja de interpretar el resto del texto hasta que encuentra literalmente la *cadena* o el carácter *EOF* (*End of File*) al comienzo de una línea y, en lugar de ello, lo envía hacia la entrada estándar del comando que vaya a ejecutar. Así, una forma de mostrar textos podría ser:

```
cat << FIN
[cuerpo del texto]
FIN
```

Las shell generan una tabla hash con los ejecutables que encuentran en el path

Con una sola línea de shell se pueden transferir jerarquías enteras de ficheros a través del mundo

REDIRECCIÓN E/S DEPENDIENTE DE SHELL

Los mecanismos de redirección son una de las características en las que difieren las dos familias de *shells* clásicas. Así pues, mientras la familia csh no permite realizar redirecciones de la salida de error, la familia sh sí. Así permite mezclar la salida de error y la estándar mediante el redirector *2>&1*. En general, *n>&m* indicará que se debe mezclar los flujos de los descriptores m y n. Así por ejemplo, *wc /bin/* > err 2>&1*, redirigirá ambas salidas STDOUT y STDERR de *wc /bin/** al fichero *err* del directorio actual o *term < /dev/modem > /dev/modem 2> /dev/null&* redirigirá STDIN y STDOUT del programa *term* al módem y STDERR a /dev/null, el recogedor de basura.

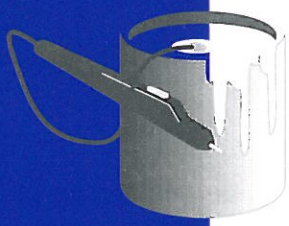
Como ejercicio sobre lo expuesto, se propone al lector que compruebe los resultados de ejecutar desde cualquier *shell*:

1. *echo in; echo competente > mi_carácter* y
2. *(echo in; echo competente) > mi_carácter*

Trate de razonar el funcionamiento interno a nivel de procesos y conexiones de descriptores en los dos casos y, si tras una relectura del artículo aún tiene dudas, podrá consultarlas a través de carta dirigida al autor a la redacción de la revista o a través de correo electrónico Internet (echeva@dit.upm.es) o Compuserve (100646,2456). En el próximo artículo, se describirá la serie de transformaciones que la *shell* realiza sobre su entrada como puente para la explicación de las estructuras de control de programación de la sh y la presentación de sus análogas en csh.

TRATAMIENTO DIGITAL DE IMÁGENES (V)

Enrique Coiras



La potencia de los ordenadores domésticos permite realizar funciones de tratamiento de imágenes que hasta hace poco estaban reservadas exclusivamente a las estaciones de trabajo y a los ordenadores dedicados. En esta quinta entrega se verán dos efectos especiales: el relieve y la mezcla de imágenes.

A partir de una imagen de grises se puede conseguir fácilmente otra que simule un *bajorrelieve* de la misma. Para ello, hay que crear la diferencia entre la imagen y una versión desplazada de la misma según un vector de traslación dado, como puede verse en el Cuadro 1. Se debe elegir bien la distancia de traslación de la imagen que se va a restar para que el efecto de relieve esté bien conseguido. Si la distancia es pequeña, el resultado se parece bastante al de cualquier operador de extracción de bordes (lo cual es normal, puesto que los operadores de extracción de bordes se basan normalmente en la diferencia de un punto y sus vecinos para hallar los contornos de las imágenes). Si es muy grande, lo que se obtiene es un caos que no tiene nada de relieve. En general, una distancia de unos cuatro *pixels* debe ser suficiente.

También es necesario elegir la dirección de traslación, ya que esto provoca el efecto que tendría la dirección de iluminación del relieve. Si se comparan los resultados del relieve generados después de trasladar la *imagen resta* según una dirección dada por un vector v = (vx, vy) y el producido al desplazarla en la dirección contraria, es decir, con un vector de traslación v = (-vx, -vy), se comprobará que son similares a los producidos al iluminar el relieve desde un lado o desde el otro, como puede verse en la Figura 1.

La rutina siguiente utiliza la técnica planteada arriba para obtener un relieve a partir de una imagen dada:

```
void relieve (imagen img, imagen
img2, int dy, int dx, imagen imgout)
{
 fimagen fimg;
 int y, x, y2, x2;

   if ((img.sizey != img2.sizey) ||
(img.sizey != imgout.sizey) || (img.sizex
!= img2.sizex) || (img.sizex !=
imgout.sizex)) {
  fprintf (stderr, "RELIEVE : Las imáge-
nes deben ser de igual tama–o!\n");
   exit (-1);
 }

crearfimagen (&fimg, img.sizey,
img.sizex);

 imagen2fimagen (img, fimg);

 for (y = 0; y < img.sizey; y ++)
  for (x = 0; x < img.sizex; x ++) {
    y2 = y + dy;
    x2 = x + dx;
   if ((y2 >= 0) && (y2 < img.sizey) &&
(x2 >= 0) && (x2 < img.sizex))
      fimg.dat [y][x] -= (float) img2.dat
[y2][x2];
 }

  normalizarf (fimg, 0.0, (float) (def-
numgrises - 1));
 fimagen2imagen (fimg, imgout);

 destruirfimagen (&fimg);
}
```

Se pueden suavizar los contornos del relieve pasándole a la imagen un filtro de desenfoque antes de aplicar la función de relieve.

El efecto de relieve también puede usarse para imágenes en color; basta con restar cada una de las componentes roja, verde y azul por separado. El relieve resultado tendrá unas variaciones de color curiosas en los bordes, debidas a las diferencias entre las componentes de cada *pixel* de la imagen en las posiciones ocupadas por los bordes y las componentes de las figuras que aparecen en ella.

En otras técnicas para conseguir simular relieves se muestrea una imagen sobre un mapa tridimensional de alturas predefinido. Éste es el típico efecto mediante el cual se realizan esas simulaciones de vuelo sobre los relieves de Marte o Venus, en las que los datos fotográficos de los planetas se superponen sobre los mapas de alturas obtenidos gracias a los análisis de radar de la superficie de los mismos.

Lo paradójico del tema es que los propios mapas de alturas pueden obtenerse de las imágenes fotográficas, siempre y cuando se tengan pares de fotos de las mismas zonas fotografiadas desde posiciones ligeramente distintas. Estos pares de fotos se llaman *pares estereoscópicos*, y son del mismo tipo que aquellas fotos o películas que se ven con gafas especiales. La diferencia es que en este caso es un ordenador el que se encarga de calcular la altura de cada punto en función de las diferencias entre las fotos. Lo que se hace realmente es seleccionar puntos significativos en una de las fotos e ir buscándolos después en la otra. Luego, como la diferencia de posición de la proyección de un punto del espacio sobre un plano depende inversamente de la distancia del punto al plano, es posible estimar las distancias de todos los puntos elegidos en función de las diferencias de

Figura 1: Relieve de una imagen con un vector de traslación v=(2,4) y con v=(-2,-4). La sombra del relieve parece venir de direcciones opuestas.

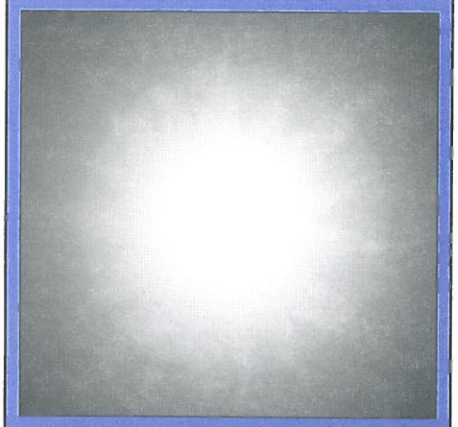


Figura 3: Imagen de una gaussiana bidimensional y el resultado de utilizarla como máscara para el fundido de dos imágenes.

FUNDIDO ENTRE DOS IMÁGENES UTILIZÁNDO LA TÉCNICA DE MEZCLA

Figura 2 : La secuencia de fotografías muestra un ejemplo práctico de encadenado de una imagen a otra variando el porcentaje de peso de la primera imagen en el resultado final en 0%, 25%, 50%, 75% y 100% respectivamente. El resultado es un *fade* entre las dos imágenes. Para porcentajes bajos la imagen final prácticamente no es distinguible, mientras que cuando son altos es la inicial la que casi no aparece.

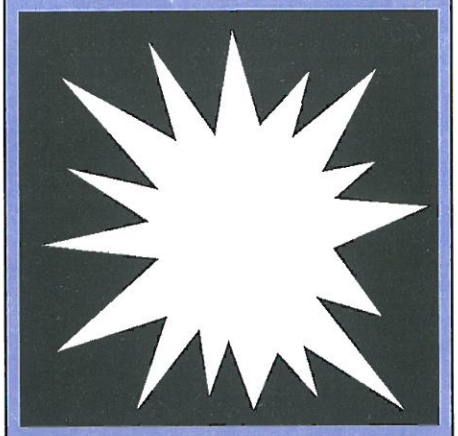
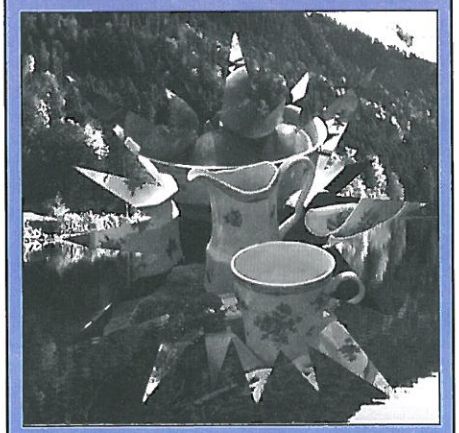
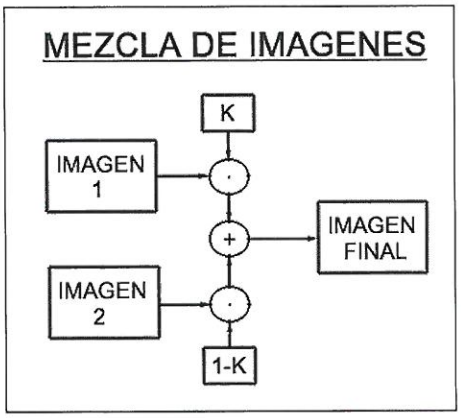


Figura 4 : Imagen binaria de un "roto" y lo que se obtiene al utilizarlo como máscara para el mezclado de dos imágenes.





Cuadro 2 : Diagrama de funcionamiento del proceso de mezcla de imágenes.

posición en cada una de las fotos.

Aunque así explicado parezca un procedimiento sencillo y las técnicas básicas para realizarlo estén claras desde hace tiempo, hay bastantes centros de investigación dedicados a este tema y a otros relacionados con él, como la visión artificial y el reconocimiento de modelos tridimensionales a partir de imágenes planas digitalizadas de los mismos.

MEZCLA DE IMÁGENES

El procedimiento de mezcla (*blending*, en inglés) tiene como objetivo representar en una misma imagen los contenidos de dos o más imágenes. La mezcla puede utilizarse para hacer "encadenados", es decir, degradados graduales que van pasando progresivamente de una imagen a otra, como se suele ver en la televisión o el cine. También se utiliza la mezcla en los procedimientos de *morphing*, para ir modificando (de forma progresiva, desde la imagen inicial a la final) los contenidos de las celdillas de distorsión.

Existen varias formas de mezclar imágenes. Una de las más primitivas consiste en calcular la suma de las dos imágenes y normalizarla para que no se salga del rango máximo de valores permitidos. De esta manera, por supuesto, no se puede dar más valor a una imagen o a la otra. Por ello, usualmente se multiplica cada imagen por una constante dada, que será mayor cuanto más peso le queramos dar a esa imagen. Las constantes deben tomar valores entre 0 y 1, y la suma de todas ellas debe ser 1. Esto se hace para mantener el rango de valores de la mezcla dentro del rango de valores inicial. La implementación de este método de mezcla, que se detalla en el Cuadro 2, puede verse en la siguiente rutina :

```
void mezclar (imagen img, imagen
img2, float pcimg1, imagen imgout)
{
  int y, x;
  float pc1, pc2, mezcla;

    if ((img.sizey != img2.sizey) ||
(img.sizey != imgout.sizey) ||
(img.sizex != img2.sizex) || (img.sizex
!= imgout.sizex)) {
    fprintf (stderr, "MEZCLAR : Las imá-
genes deben ser de igual tama–o!\n");
    exit (-1);
  }

  pc1 = pcimg1 / 100.0;
  pc2 = 1.0 - pc1;

  for (y = 0; y < img.sizey; y ++)
    for (x = 0; x < img.sizex; x ++) {
     mezcla = (float) img.dat [y][x] * pc1
+ (float) img2.dat [y][x] * pc2;

      imgout.dat [y][x] = (unsigned char)
mezcla;
    }
}
```

El valor de *pcimg1* controla el porcentaje de la primera imagen que se añadirá a la imagen resultado, y *100.0 - pcimg1* es, evidentemente, el porcentaje de aportación de la segunda imagen. Si *pcimg1* vale 100.0 la imagen final será idéntica a la primera imagen, y si vale 0.0 será igual que la segunda.

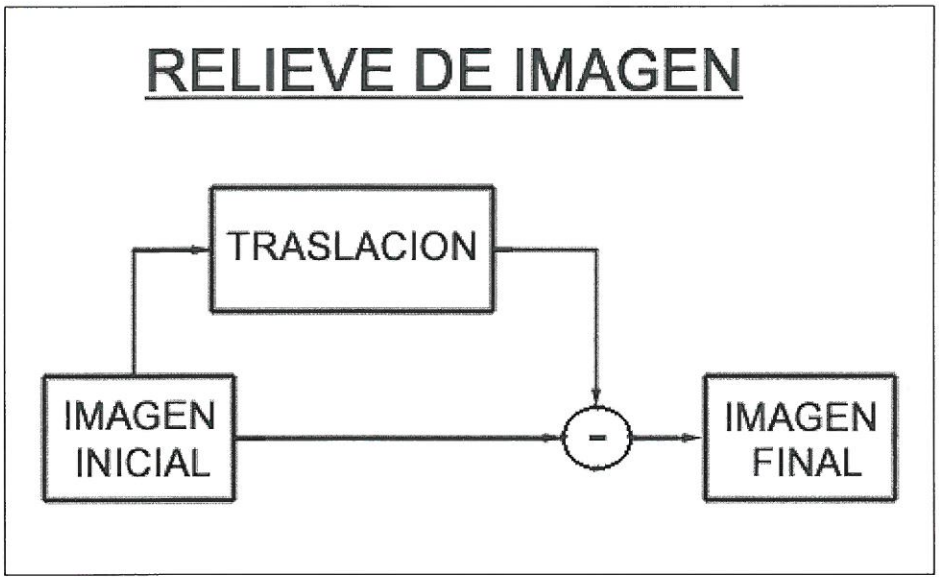
No se debe normalizar, en general, el resultado de la mezcla, pues se producirían variaciones en el brillo de la imagen final que serían apreciables en una secuencia de encadenado. Si no se normaliza, se asegura que los extremos de la secuencia se corresponderán con las dos imágenes de entrada elegidas.

La dirección de traslación se corresponde con la de iluminación del relieve

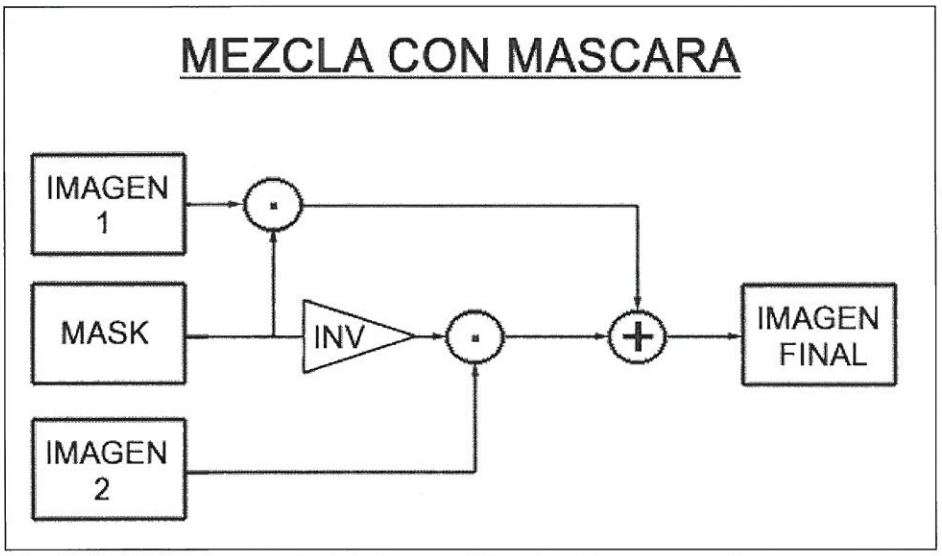
En la Figura 2 se puede ver cómo la variación de las constantes cambia el resultado de la mezcla final para dos imágenes distintas, dando lugar a un encadenado.

Este tipo de procedimiento de mezcla también puede usarse para encadenar imágenes en color, mezclando por separado cada una de las componentes.

La mezcla de imágenes también puede utilizarse para combinar varias imágenes de una escena que no contengan exactamente la misma información. Por ejemplo, para encadenar las distintas bandas de frecuencias de



Cuadro 1 : Fundamento de la técnica de simulación de relieve de imágenes.



Cuadro 3 : Proceso de mezcla de imágenes utilizando máscaras como funciones peso.

las imágenes tomadas por satélites, o para comparar fotografías tomadas en instantes temporales diferentes y detectar variaciones en ellas. En estos casos, sin embargo, el procedimiento de mezcla descrito aquí puede no ser el más indicado. Lo ideal en esos casos es que los puntos de las imágenes tengan más o menos peso según la cantidad de información que aporten a la imagen final. Para asignar esos pesos hay que utilizar funciones locales que los estimen, como la varianza, por ejemplo, que daría más valor a los puntos que se diferenciasen de sus vecinos más de lo normal. Así se evitaría que las zonas constantes de una de las imágenes redujeran la discernibilidad de la otra imagen en dichas zonas, sin aportar ninguna información útil.

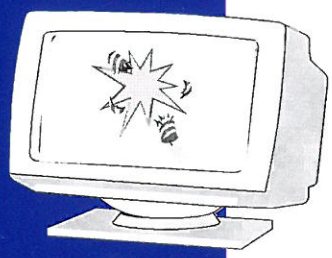
Otra forma de mezclar imágenes se basa en la utilización de máscaras para controlar qué partes de cada una de las dos imágenes de entrada se destacarán más o menos. El procedimiento de mezcla visto antes sería un caso particular del mezclado con máscaras en el que la máscara es una imagen uniforme de valor constante 1/2. La siguiente función toma tres imágenes como entrada (las dos imágenes a mezclar y otra más con la máscara) y su modo de operación puede verse en el diagrama del Cuadro 3. La máscara deberá ser más clara en las zonas en las que se quiera que la primera imagen tenga más peso. La segunda imagen será modificada por una máscara inversa a la especificada; así se consigue que la suma de las dos imágenes multiplicadas por sus máscaras respectivas se mantenga dentro de los límites de valores permitidos por nuestro rango de grises. La rutina es la siguiente:

```
void mezclar_mascara (imagen img, imagen img2, imagen mask, imagen imgout)
{
 int y, x;
 float pc1, pc2, mezcla, normfactor;

   if ((img.sizey != img2.sizey) || (img.sizey != imgout.sizey) || (img.sizey != mask.sizey) || (img.sizex != img2.sizex) || (img.sizex != imgout.sizex) || (img.sizex != mask.sizex)) {
  fprintf (stderr, "MEZCLAR_MASCARA : Las imágenes deben ser de igual tama-o!\n");
   exit (-1);
 }

 normfactor = 1.0 / (float) (defnumgrises - 1);

 for (y = 0; y < img.sizey; y ++)
  for (x = 0; x < img.sizex; x ++) {
   pc1 = (float) mask.dat [y][x] * normfactor;
    pc2 = 1.0 - pc1;
   mezcla = (float) img.dat [y][x] * pc1 + (float) img2.dat [y][x] * pc2;

    imgout.dat [y][x] = (unsigned char) mezcla;
  }
}
```

En la Figura 3 se puede ver cómo la utilización de una máscara con perfil *gaussiano* genera una imagen mezcla que tiene mayor presencia de la primera imagen cerca del centro, mientras que los bordes están prácticamente ocupados por la segunda imagen. Si la máscara tiene bordes abruptos, como en el caso de la Figura 4, la imagen final será una superposición de la primera imagen sobre la segunda en las zonas en las que la máscara lo indique.

EL PLASMA

Pedro Antón



Sobre el plasma no hay nada escrito, a cada programador se le puede ocurrir su propio plasma, aquí se intentará explicar el plasma que Spanish Lords ha usado en sus demos.

En este artículo se va a explicar cómo se realiza un efecto "demoero" en el que se dibuja una serie de curvas senoidales asociadas y en movimiento. Visto así, puede parecer algo extraño, pero el mecanismo para realizar el efecto del plasma es bastante simple. En este efecto, de nuevo adquiere una elevada importancia el concepto del buffer, y veremos llevada hasta una efectividad asombrosa el uso de las tablas con datos precalculados.

El denominado efecto del plasma se divide en dos fases: dibujo del plasma original y movimiento circular mezclando el plasma ya creado.

DIBUJO DEL PLASMA ORIGINAL

Aquí es donde adquieren importancia las tablas de datos precalculadas. En esta fase, se prepara un dibujo de curvas senoidales que será el que posteriormente se mezclará. Este dibujo será de una dimensión mayor a la que se mostrará, si se va a trabajar con una resolución de 160 x 100 (como en el caso del código fuente incluido), el buffer debe ser mayor de dicha resolución (en el código es de 320 x 200).

El fundamento matemático del plasma se basa en el dibujo de circunferencias concéntricas, de tal forma que la variación del color de las circunferencias depende directamente del radio de éstas. Para saber el radio de una circunferencia que pasa por un punto (x,y), se aplicará el teorema de Pitágoras: En un triángulo rectángulo, la hipotenusa al cuadrado es igual a la suma de los cuadrados de los catetos. Pero como una circunferencia es el lugar geométrico de los puntos del plano que equidistan de otro, llamado centro, una distancia fija llamada radio, la hipotenusa será el radio (ver figura 1).

Pero este dibujo quedaría extremadamente simple, es tan sólo el dibujo de unas circunferencias, aunque es la base para realizar todas las modificaciones que posteriormente nos darán el efecto del plasma buscado. A la fórmula ya descrita, se le pueden aplicar una o más curvas senoidales, de forma que en el dibujo original no queden sólo circunferencias concéntricas, sino que dichas circunferencias se agrupan formando otras más pequeñas. Un ejemplo de una fórmula con curvas senoidales es el usado en el código que acompaña el artículo:

```
Color = Sin((Sqrt(CntX2+CntY2))*4)*16 + Cos(CntX)*32
```

No obstante, el lector puede aplicar la fórmula que crea conveniente para generar las modificaciones adecuadas al dibujo original.

LAS TABLAS

Con este tipo de cálculos se hace totalmente necesario el uso de tablas para poder generar el dibujo inicial en tiempo real, o al menos en un tiempo aceptable. Visto el tipo de operaciones que se deben hacer, tales como la raíz cuadrada, el seno y el coseno, es evidente que los valores con los que se va a trabajar son valores decimales, no enteros, por lo que esto se traduciría en un consumo de tiempo grandioso, si se usara coma flotante. Para solucionar este problema se usará el punto fijo en aquellas partes de cálculo que lo requieran.

En el cálculo de las raíces cuadradas no se usa punto fijo, ya que no será necesario, pues se calculará la raíz cuadrada de números relativamente grandes, obteniendo así un número bastante más pequeño, por lo que la precisión decimal no se hace necesaria, puesto que la oscilación mayor

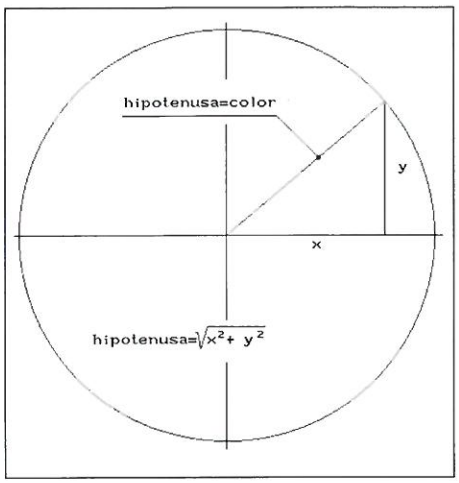


Figura 1. Cálculo del radio por el teorema de Pitágortas.

entre dos valores precalculados es la unidad. Si fuese el caso inverso, el cuadrado de un número decimal, la variación entre el cuadrado de un número y el siguiente puede hacerse muy grande, en el caso de precisar decimales:

$(200.7)^2 = 40280.49$
$(200\ )^2 = 40000$

La variación de la raíz cuadrada no es superior a la unidad en los intervalos que se usarán, mientras que la variación del cuadrado sí podría serlo. Dicho de otra manera, decir que la raíz cuadrada de 40280 es 200 es más aceptable que decir que el cuadrado de 200.7 es 40000.

Visto que la precisión decimal de la raíz cuadrada no es muy importante en este caso, se pasará a explicar el método de cálculo que se usa en el código que acompaña la revista. En este caso se crea una tabla con el cuadrado de los números del 0 al 640. Para almacenar estos números se usan 32 bits, puesto que a partir de 255, el cuadrado ocupa más de 16 bits. Con esta tabla ya se puede calcular el cuadrado de dichos números sin necesidad de multiplicar. Pero para el caso de la raíz cuadrada, el proceso es un poco más complejo.

En el caso de la raíz cuadrada, dado un valor, se debe buscar entre qué dos intervalos de la tabla se encuentra dicho valor. Para esto se usa el algoritmo de búsqueda binaria, calculando así de la forma más rápida posible el valor de dicha raíz cuadrada.

EL ALGORITMO DE BÚSQUEDA BINARIA

Mediante este algoritmo, el procedimiento que calcula la raíz cuadrada tiene como parámetros de entrada el valor del número cuya raíz cuadrada queremos hallar y el valor mínimo y máximo que podrá tener dicha raíz cuadrada. Cuanto menor sea el intervalo entre estos dos valores, más rápido será el cálculo de la raíz cuadrada. Esto, en líneas generales, en el caso del ejemplo los valores mínimo y máximo son siempre 0 y 380 respectivamente.

```
{ FUNCTION FastSquareTable.          }
{ Devuelve la raíz cuadrada de Dato. }

 Function FSqrt(Min,Max:Word;
Dato:LongInt):Word;

 Label
  Vuelve,Fin;
 Var
  Cursor : Word;
  AuxVal : LongInt;

 Begin
 Vuelve:
 Asm
  mov  ax,Max
  add  ax,Min
  shr  ax,1
  mov  Cursor,ax
 End;
  AuxVal:=SquareTable[Cursor];
  If (AuxVal=Dato) or (Max=Min+1)
then goto fin;
  If AuxVal>Dato then Max:=Cursor;
  If AuxVal<Dato then Min:=Cursor;
  GoTo Vuelve;
Fin:
  FSqrt:=Cursor;
end;
```

El funcionamiento de este algoritmo es simple:

- Se divide el intervalo de números en dos intervalos simétricos.
- Se compara el número dado (del cual se va a hallar la raíz) con el valor almacenado en el punto que comparten los dos intervalos, es decir, el punto medio del intervalo inicial.
- Si el valor dado es mayor que el existente en ese punto, entonces el valor menor se iguala al valor medio, y se repite el proceso de nuevo.
- Si el valor dado es menor que el almacenado en el valor medio, entonces el valor mayor se iguala al valor medio, y se repite el proceso de nuevo.

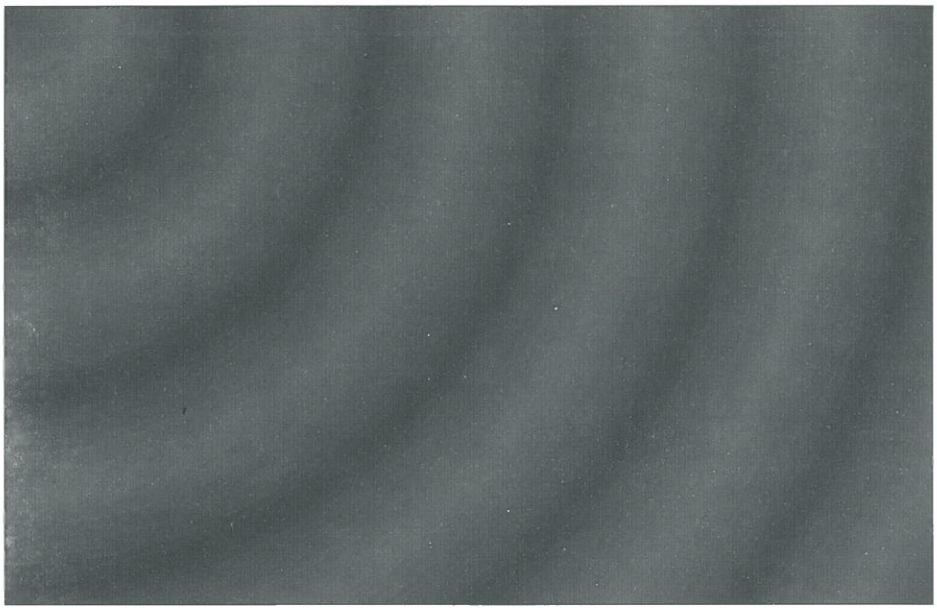
Este proceso finalizará cuando el intervalo sea igual a uno, es decir, que la raíz cuadrada pueda tomar dos valores, la raíz del valor menor y la raíz del valor mayor.

En este momento se puede redondear hacia arriba o hacia abajo, tomando como valor final el valor menor o el valor mayor. En el código fuente empleado en la revista, se redondea hacia abajo, tomando el valor menor.

Este algoritmo es la puesta en práctica del viejo refrán: "DIVIDE Y VENCERÁS". Así, el funcionamiento de este algoritmo consiste en ir subdividiendo el intervalo inicial en dos intervalos más pequeños. En este momento se comprueba si la raíz cuadrada que queremos hallar está en el intervalo superior o en el intervalo inferior. De esta forma, se va subdividiendo el resultado hasta que no se pueda subdividir más, y es en ese momento cuando se ha hallado la raíz cuadrada.

Con esto finalizaría la optimización del procedimiento que se encarga de dibujar el plasma original. Este proceso se suele realizar en tiempo no real, es decir, se crea el plasma con una precisión mayor y sin hacer uso de las tablas, y después se almacena, para en el momento de mezclar y mover el dibujo, cargarlo desde disco en lugar de generarlo. En el caso de *Spanish Lords*, siempre hemos generado el plasma en tiempo real, lo que consume unos dos segundos en un 486DX 33, ese tiempo muerto puede usarse para mostrar una gran creación artísti-

Las raíces cuadradas también pueden precalcularse



Aspecto del plasma de círculos concéntricos.

```
  mov   CntY,100
@BucleY:
  mov   CntX,160
@BucleX:
  mov   si,X1
  mov   ax,[si]
  mov   si,X2
  add   ax,[si]
  mov   si,X3
  add   ax,[si]
  mov   si,X4
  add   ax,[si]
  and   al,3Fh
  mov   ah,al
  mov   es:[di   ],ax
  mov   es:[di+320],ax
  add   di,2
  inc   X1
  inc   X2
  inc   X3
  inc   X4
  dec   CntX
  jnz   @bucleX
  add   di,320
  add   X1,160
  add   X2,160
  add   X3,160
  add   X4,160
  dec   CntY
  jnz   @BucleY
```

Listado 1. Bucle principal rellenar el *buffer*.

ca, léase dibujo, de un grafista del grupo ¿no?

EL BUFFER. MEZCLA DEL PLASMA

Esta es la fase verdadera del efecto del plasma. Aunque no se puede realizar sin la primera, la parte que dibuja el plasma sí puede ser realizada en otro momento, pero ésta no. Esta fase debe estar altamente optimizada, puesto que es la parte que consume el 100% del tiempo del microprocesador. Se debe resaltar la atención del lector en este detalle, ya que si el efecto consume el 100% de procesador, ¿qué diferencia habrá entre dos equipos de muy distintas velocidades (486DX 33 - Pentium 90)? Realmente mucha, optimizando mucho en un 486DX 33 se pueden conseguir unos 35 frames por segundo, y en un Pentium la velocidad vendrá limitada por el retrazo vertical, es decir, 70 frames por segundo, y la diferencia es realmente importante, además a 70 frames este efecto no resulta bonito. En posteriores artículos veremos cómo solucionar este problema.

Hay que tratar de evitar que el plasma parezca repetirse

Esta fase consiste en mezclar distintas partes del dibujo del plasma ya creado y visualizar el resultado, el funcionamiento es bastante simple. Se escogen cuatro partes distintas dentro del plasma original; en este caso, como el plasma original es de 320 x 200, se cogen cuatro partes de 160 x 100. Así, se suman los valores de cada parte y se calcula la media aritmética. Los puntos resultantes formarán el buffer que vamos a volcar a la memoria de vídeo. Esto se verá mejor gráficamente en la figura 2.

Esto, a simple vista, no parece complicado, pero cuando se quiere conseguir código rápido se plantea un problema. Por cada pixel hay que sumar y dividir, y si la pantalla va a ser de 160 x 100, se sumará 16000 veces por frame, algo que se paga en velocidad si el bucle principal no está optimizado.

Por cierto, se debe añadir una breve nota al programa del ejemplo. El buffer original está hecho en 320 x 200, y cuando se mezcla, a 160 x 100. Pero el buffer que se vuelca a vídeo es de 320 x 200, luego será necesario escribir 2 x 2 pixels en el buffer por cada uno de la mezcla, como ya se hizo en el fuego. Más adelante se verá cómo se encargará la VGA de poner las segundas líneas, ahorrando así bastantes ciclos.

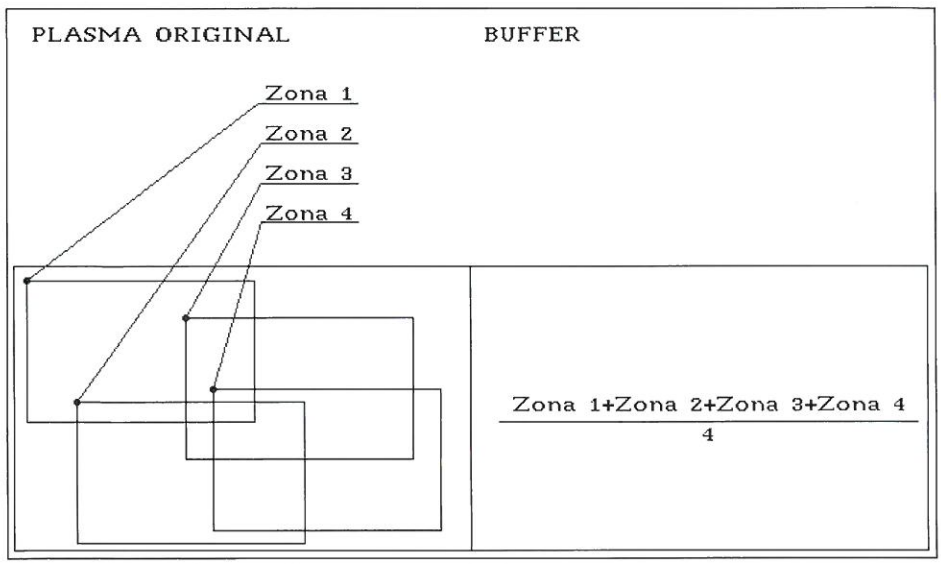
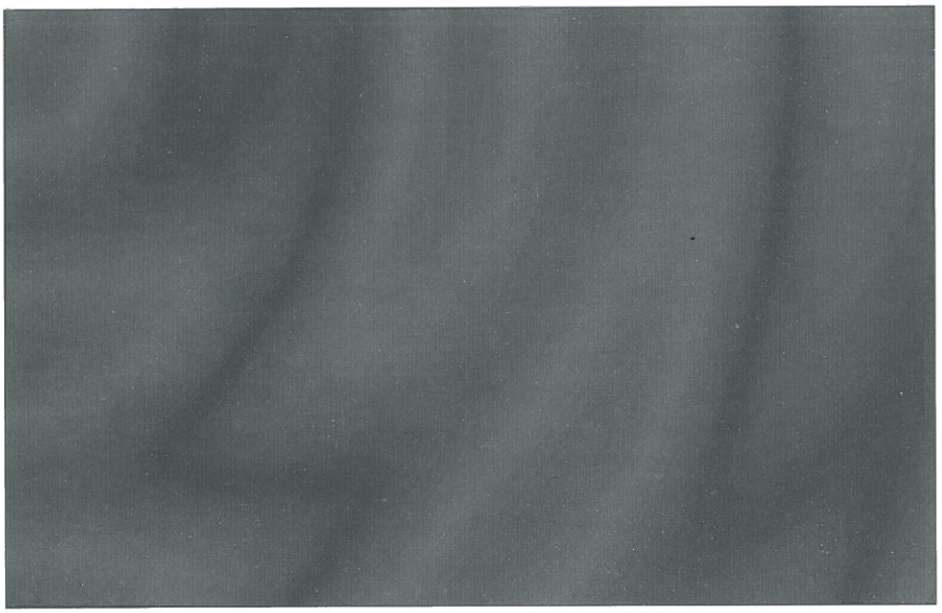


Figura 2. Utilización del buffer para mostrar el plasma.



Vistosa variación del efecto del plasma circular muy utilizada en las demos.

EL BUCLE PRINCIPAL

El bucle principal es el encargado de realizar la media aritmética de los cuatro puntos que se escogen para calcular el plasma, es decir, el código que se encarga de rellenar el buffer que luego se volcará a vídeo:

En el *BucleX* es donde se realiza la verdadera mezcla. El *BucleY* se encarga únicamente de ajustar las medidas del plasma en 320 x 200.

Como se ve en este fragmento de código, se usan muchas variables auxiliares, tales como *X1,X2,X3,X4,CntX* y *CntY*, y eso se traduce en una pérdida de tiempo. Esto quiere decir, que usar este tipo de variables en memoria es mucho más lento que usar directamente los registros del procesador, por lo que si se quiere optimizar altamente el bucle, se debe evitar el uso de dichas variables; en el caso del ejemplo es mucho más fácil para el lector ver el funcionamiento del código de esta forma.

Conviene destacar la función de *AND AL,3Fh*; limitar el número de colores que está usando el plasma a 64 colores, porque cuando se diseñó por primera vez este plasma, se pensaba superponer un dibujo que pudiese usar los otros 196 colores, pero en fin, eso no nos interesa en este artículo. Para cualquier lector sería una buena experiencia intentar que esto funcionase con 256 colores, aunque "tal vez" tenga que considerar algo sobre la paleta y sobre el dibujo del plasma original.

```
X1:=(80*FSin(Circle3) div 256)+80;
Y1:=(50*FCos(Circle3) div 256)+50;
X2:=(80*FSin(Circle1) div 256)+80;
Y2:=(50*FCos(Circle1) div 256)+50;
X3:=(80*FSin(Circle2) div 256)+80;
Y3:=(50*FCos(Circle2) div 256)+50;
X4:=(80*FSin(Circle4) div 256)+80;
Y4:=(50*FCos(Circle4) div 256)+50;
X1:=Y1*320+X1;
X2:=Y2*320+X2;
X3:=Y3*320+X3;
X4:=Y4*320+X4;
Circle1:=Circle1+Add[1];
Circle2:=Circle2+Add[2];
Circle3:=Circle3+Add[3];
Circle4:=Circle4+Add[4];
```

Listado 2. Código para el movimiento.

EL MOVIMIENTO

Hasta aquí se ha visto cómo se dibuja un solo frame del plasma, pero ¿cómo hacer que el dibujo cambie? Es un procedimiento bastante simple. Consiste en ir variando la posición de esas cuatro partes del plasma original que se capturan. Así, no sólo se produce un movimiento por el plasma original, sino que se producen cuatro movimientos distintos, y la variación no es simple, sino que se convierte en un movimiento variado que no parece un simple *scroll circular* por el plasma original.

Para producir este movimiento dentro del plasma original, se mueve el origen de los cuatro buffers a mezclar, de forma circular, usando las tablas de senos y cosenos. Se mantienen unas variables de ángulo que serán incrementadas con unos valores determinados para cada frame. Así, las variables de ángulo, en el ejemplo serían las variables *Circle*, y las variables que se suman a dichos ángulos son las que se encuentran en el array *Add*.

El código del ejemplo que realiza el movimiento se muestra en el listado 2.

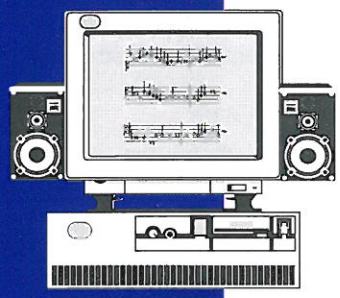
LA PALETA

Y por último, pero no por ello menos importante, se encuentra el apartado de la paleta. Se debe decir que es el alma del efecto. Si la paleta no es la correcta, el efecto puede quedar francamente feo, pero si la paleta es la adecuada, se puede multiplicar la belleza del plasma. En el caso del ejemplo, se incluye una paleta que no varía, pero se puede implementar la variación de la paleta en tiempo real, de forma que, ateniéndose a unas reglas de creación de paleta, el paso de unos colores a otros sea progresivo, y no se aprecien los cambios de ésta cada vez que se produzcan, sino que todo se integre en un mismo proceso, que cambie la forma del dibujo a la vez que sus colores.

Sobre la paleta, hay que observar dos reglas fundamentales. La primera de ellas es que se deben tener en cuenta los colores en los que va a oscilar el plasma dibujado. En el caso del ejemplo entre 0-64 (atención a aquellos lectores que intenten variar el ejemplo a 256 colores). La segunda regla es que el color con el que empieza dicho intervalo debería ser el color con el que acaba para que el efecto sea el deseado. De todo esto se deduce que el intervalo se puede dividir en dos partes, y en las dos incluir la misma paleta en orden inverso.

La paleta es una de las partes más flexibles de este efecto. Cada lector puede intentar conseguir la suya propia, que puede variar considerablemente la apariencia del efecto.

Y esto es todo sobre nuestro plasma, ¿serás tú capaz de hacer uno distinto?, ¿si?, pues adelante.



REPRODUCCIÓN EN ESTÉREO Y 16 BITS

Luis Crespo

En la segunda entrega de esta serie de artículos (número 5 de la revista) se "visitaron" las tarjetas de la serie Sound Blaster, explicando las características y diferencias entre los distintos modelos. Se presentó una serie de rutinas para reproducir digitalizaciones usando el DMA y la técnica del doble *buffer*. Este mismo sistema se utiliza para reproducir digitalizaciones estéreo y a 16 bits. Como se parte de esa base para ampliarla, sería conveniente echar un vistazo a ese artículo para refrescar las ideas, y tenerlo a mano mientras se lee éste.

Con las entregas anteriores ya se puede reproducir cualquier fichero MOD soportando la mayoría de los efectos... pero en mono y a 8 bits. Es el momento de aumentar en gran medida la calidad sonora de nuestro reproductor, haciéndolo sonar en estéreo y a 16 bits si es que el hardware lo permite.

ESTÉREO EN LA SOUND BLASTER

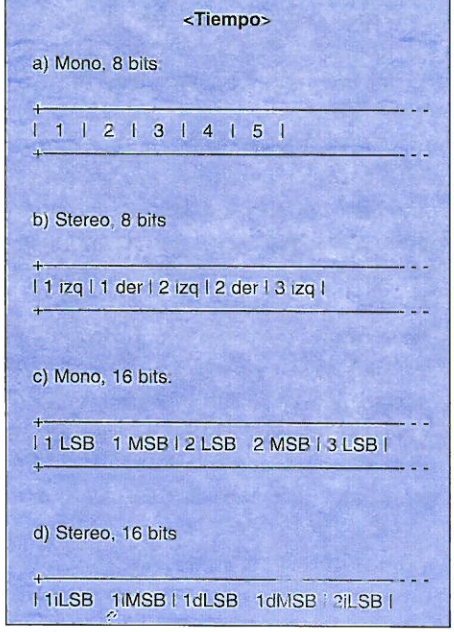
¿Cómo se envía una digitalización estéreo a la SB? Una digitalización estéreo no es ni más ni menos que dos digitalizaciones que deben reproducirse simultáneamente: una para el canal izquierdo y otra para el derecho. Como deben sonar al mismo tiempo, no puede enviarse primero una y luego otra, de modo que se alternan las muestras del canal izquierdo y del derecho (ver figura 1). Por convenio se envía primero la muestra del canal izquierdo y luego la del derecho.

Recordemos que para reproducir una digitalización en la SB, tras indicar a la tarjeta la frecuencia de muestreo y tamaño del *buffer*, enviando un comando determinado al puerto *Base+0Ch* se iniciaba la transferencia, de modo que la digitalización empezaba a sonar. El byte de comando varía según la versión del DSP del que se disponga, o lo que es lo mismo, del modelo de SB.

El byte de comando utilizado en la SB 16 para la reproducción de digitalizaciones de 8 bits es el *C6h* seguido de un byte que indica la modalidad, y de dos bytes que indican el número de muestras a transferir menos 1. Lo que nos interesa es la modalidad, ya que indica el tipo de datos que se están transfiriendo. Si el bit 5 de este byte está a 0, la transferencia es mono, y si está a 1 es estéreo. Si el bit 4 está a 0, las muestras son sin signo, si está a 1, con signo. El resto de bits deben estar a 0. Nos interesa en este caso una transferencia en estéreo y sin signo (sin signo por compatibilidad con versiones más antiguas de SB). Como el número de muestras se ha duplicado, tras el byte de modo los bytes de tamaño tendrán en cuenta esta cuestión. También hay que tener en cuenta esta cuestión a la hora de programar el controlador de DMA y de reservar memoria para el *buffer*.

Hagamos un pequeño paréntesis: ¿qué diferencia hay entre muestras con signo y sin signo? En el primer caso, la muestra de amplitud nula (si es que se puede

FIGURA 1


<Tiempo>
a) Mono, 8 bits
| 1 | 2 | 3 | 4 | 5 |
b) Stereo, 8 bits
| 1 izq | 1 der | 2 izq | 2 der | 3 izq |
c) Mono, 16 bits.
| 1 LSB | 1 MSB | 2 LSB | 2 MSB | 3 LSB |
d) Stereo, 16 bits
| 1iLSB | 1iMSB | 1dLSB | 1dMSB | 2iLSB |

Disposición de las muestras según el modo de reproducción.

decir que una muestra tiene amplitud) es la que vale 0, y las de máxima amplitud son el +127 y el -128, por arriba y por abajo de la posición central. En el caso de una muestra sin signo, los valores que puede tomar son de 0 a 255, con el centro o posición de amplitud nula en el 128, y los extremos o posiciones de máxima amplitud en el 255 y el 0. Estos valores son, por supuesto, en el caso de muestras de 8 bits. Sólo la SB 16 y modelos superiores permiten la transferencia de muestras con o sin signo; todos los modelos anteriores sólo aceptan muestras sin signo, con el punto central en el 128.

Si se tienen en cuenta estas cuestiones, en el caso de la SB 16 no es necesario ningún tratamiento adicional para conseguir sonido estéreo. El punto clave es que las muestras de ambos canales deben estar entrelazadas o intercaladas: canal izquierdo, canal derecho, canal izquierdo, canal derecho, etc.

La SB Pro se programa de forma un tanto distinta: se utiliza una especie de "truco". Hay un bit en los registros del mezclador que indica el modo estéreo, pero el DSP se programa como si se tratase de una transferencia mono. Como se envían el doble de muestras, esto no sólo influye en el tamaño del *buffer* de transferencia, sino que la frecuencia de muestreo se duplica. Esto es así porque el DSP ignora que la transferencia es estéreo, y debe reproducir el doble de bytes en el mismo periodo de tiempo. En el caso de la SB 16 esto no ocurre porque el DSP es consciente de la modalidad y por lo tanto del número de bytes que debe leer en cada ciclo de DMA.

Todas estas distinciones y comprobaciones se llevan a cabo en las rutinas *ProgDSP* y *SBPlayBuf* del fichero *SB.PAS*, si bien hay que tener en cuenta que el tamaño del *buffer* que recibe la rutina *SBPlayBuf* viene dado en bytes y no en muestras, con lo que se ahorran algunos cálculos.

16 BITS EN LA SB 16

La SB 16 tiene un DAC de 16 bits, lo cual permite una gran calidad en la reproducción de sonido si se utiliza adecuadamente. Una digitalización a 16 bits ocupa el doble que su equivalente de 8 bits a una frecuencia determinada; para agilizar la transferencia del PC a la tarjeta, la SB 16 permite la utilización de un canal de DMA de 16 bits, lo cual reduce la carga del bus, transfiriendo el doble de información en un mismo ciclo.

A partir del PC AT, todos los PCs incorporan 4 canales de DMA de 16 bits aparte de los 4 canales de 8 bits, de modo que programando convenientemente el controlador de DMA de 16 bits se puede conseguir que la SB 16 reproduzca digitalizaciones de 16 bits. Pero esta cuestión puede dar lugar a problemas: algunas placas de PC no son compatibles con la SB 16 usando el DMA de 16 bits (o viceversa), e intentar usarlo da lugar a errores de paridad de memoria y lógicamente a cuelgues del ordenador. De modo que la SB 16 permite la posibilidad de reproducir digitalizaciones de 16 bits por un canal de 8 bits. El usuario de la SB 16 suele configurar sus IRQs y sus canales de DMA en el arranque, y la rutina *SBDetect* detecta sus valores leyéndolos de los registros adecuados. Si el canal de DMA de 16 bits está a 0, esto indica que debe utilizarse el mismo canal que el de las transferencias de 8 bits. De todas formas, la tarjeta se programa del mismo modo tanto en un caso como en el otro.

Para programar la SB 16 de forma que reproduzca muestras de 16 bits se utiliza el comando *B6h*, que tiene un comportamiento idéntico al *C6h*: tras el comando se envía un byte en el que se indica el modo: si las muestras son con signo o sin signo, y si se trata de una digitalización estéreo o mono. Y tras el byte de modo se envía por el mismo puerto el número de muestras menos 1, primero el byte bajo y luego el alto. Como una muestra de 16 bits ocupa 2 bytes, hay que tener en cuenta esto a la hora de reservar memoria para el *buffer* y programar el KDMA. En el *buffer* de la memoria del PC, las muestras de 16 bits deben disponerse como se disponen todos los números de 16 bits en memoria en la arquitectura Intel: el byte bajo en la dirección baja y el byte alto en la dirección alta. Los 16 bits y el estéreo pueden combinarse sin problemas, siempre teniendo en cuenta que una digitalización estéreo tiene el doble de muestras que una equivalente en mono (ver figura 1).

EL DMA DE 16 BITS

La rutina *ProgKDMA* del fichero *HARDWARE.PAS* ha sido ampliada para soportar los canales de DMA de 16 bits. Los canales de 8 bits van del 0 al 3, y los de 16 bits van del 4 al 7, así que según el número del canal la rutina determina automáticamente si se trata de una transferencia de 8 o 16 bits. El protocolo de programación del DMA de 16 bits es muy similar al de 8 bits, que se explicó en el artículo del número 5. Las principales diferencias son los puertos en los que se mapea el controlador y el sistema de direccionamiento de memoria física.

El puerto *D8h* es el registro de máscara individual y se utiliza para habilitar y deshabilitar cada canal de DMA. En los dos bits más bajos se coloca el número de canal (como los canales de 16 bits van del 4 al 7 hay que restar 4 al número de canal), y en el bit 2 se coloca un 1 para deshabilitar el canal indicado o un 0 para habilitarlo. Es necesario deshabilitar un canal durante su programación, al igual que las interrupciones, o de lo contrario los resultados pueden ser inesperados.

El puerto *D6h* es el registro de modo, y se utiliza para especificar el tipo de transferencia para cada canal. Funciona de la misma forma que el registro de modo del KDMA de 8 bits: los dos bits de menor peso indican el número de canal que se va a programar (de 0 a 3: de nuevo hemos de restar 4). Los bits 2 y 3 indican el sentido de la transferencia: 01 para una transferencia de periférico a memoria, y 10 para una transferencia de memoria a periférico. El bit 4 habilita el modo auto-inicializado: si vale 1 el modo auto-inicializado está activado, si vale 0, no. El bit 5 indica si la dirección de memoria debe incrementarse o decrementarse tras cada transferencia, y por último los bits 6 y 7 indican la modalidad de transferencia y deben valer 1 y 0 respectivamente.

Los puertos *C2h*, *C6h*, *CAh* y *CEh* se utilizan para indicar el tamaño del bloque a transferir para el canal 4, 5, 6 y 7 respectivamente. Como el KDMA de 16 bits trabaja con *words*, el tamaño que se indica es en *words*. Además, el tamaño se especifica con un *word*, con lo que es necesario enviar primero el byte bajo y luego el alto por el mismo puerto. Para

asegurarse que el KDMA está listo para recibir el byte adecuado hay que escribir cualquier valor en el puerto D8h. De esta forma se indica que el siguiente byte que se escriba será el bajo, el siguiente el alto, el siguiente el bajo, etc. Hay que tener en cuenta que el número que se escribe en estos puertos es el número de *words* a transferir menos uno, de forma que, por ejemplo, para transferir 100 bytes hay que escribir un 49.

Al igual que con el DMA de 8 bits, existen puertos en los que se indica la dirección física de la memoria del PC desde donde se iniciará la transferencia (o hacia donde en el caso de lectura). Como en un AT la máxima memoria direccionable son 16 Mb, se necesitan 24 bits para especificar una dirección. Pero al transferirse datos de 16 bits, las direcciones de memoria se dan de la siguiente forma: el bit 0 de los 24 bits de dirección se considera que siempre es 0, de modo que los datos deben estar alineados en direcciones pares.
Los bits 1 al 16 se colocan en el puerto de dirección base, y los bits 17 a 23 se colocan en los bits 1 a 7 del registro de página.

Los puertos de dirección base para los canales 4 5 6 y 7 están en el *C0h*, *C4h*, *C8h* y *CCh* respectivamente. Como reciben un *word*, de nuevo hay que suministrar primero el byte bajo y a continuación el alto. Y los registros de página para los canales 4, 5,6 y 7 están en el *8Fh*, *8Bh*, *89h* y *8Ah* respectivamente. Recordemos que una dirección física puede calcularse con la siguiente fórmula a partir de una dirección lógica:

*Dirección Física = Segmento * 16 + Offset*

Siempre y cuando el procesador esté en modo real o virtual. También sigue ocurriendo que un bloque no puede estar solapado en dos páginas, con lo que habrá que comprobar que esto no ocurre al reservar memoria. Para ello nos servirá la misma rutina utilizada para reservar un *buffer* de DMA de 8 bits. Como ahora las páginas son de 128 Kb en lugar de 64 Kb, esta rutina encontrará solapamientos donde quizás no los haya, pero de esta forma se puede utilizar para *buffers* tanto de 8 como de 16 bits.

ESTÉREO EN EL MOD PLAYER

Ahora que ya sabemos cómo reproducir digitalizaciones estéreo y de 16 bits, sólo nos falta saber cómo debe generarlas nuestro reproductor de MODs. De hecho las rutinas de mezcla ya trabajan con datos de 16 bits para tener más precisión, de forma que sólo habrá que modificar la rutina que llena el *buffer* de DMA a partir del *buffer* en el que se mezclan las 4 o más voces del MOD, para que no descarte los 8 bits bajos de cada dato como se hacía anteriormente.

Para poder generar una salida estéreo se necesitarán dos *buffers* de mezcla: el izquierdo y el derecho. Así, las rutinas de mezcla, para cada voz del MOD tendrán en cuenta si debe sonar por la izquierda o por la derecha, y la mezclarán en el *buffer* correspondiente. Estos dos *buffers* están intercalados, es decir, un *word* del izquierdo, uno del derecho, etc., para facilitar el posterior llenado del *buffer* de DMA.

Durante la carga del fichero se calculan los volúmenes izquierdo y derecho de cada voz y se almacenan en una estructura de datos. Esta estructura es lo bastante genérica para permitir que una voz suene al mismo tiempo por la izquierda y por la derecha, y a distintos volúmenes. Pero de momento no nos complicaremos la vida y sólo consideraremos como válido el lado que tenga mayor volumen. Para ello, se ha añadido un nuevo parámetro a la rutina *FillBuffer* del fichero *PLAY.PAS*: un *booleano* que indica si hay que mezclar esa voz en el *buffer* izquierdo o el derecho. Si quisiéramos soportar el caso más general de voces con volumen izquierdo y derecho ambos distintos de 0, la rutina de mezcla se complicaría mucho y consumiría mucho tiempo (habría que hacer dos multiplicaciones por muestra y empezarían a faltarnos registros del procesador). Además, sólo tenemos en cuenta el volumen global, ignorando el valor del volumen de cada voz, al igual que en la entrega anterior.

Tan sólo queda llenar convenientemente el *buffer* de DMA a partir de los *buffers* izquierdo y derecho de mezcla de voces del *player*. Como hay muchas formas posibles de hacerlo (8 o 16 bits, mono o estéreo), se utiliza un puntero a la rutina de llenado, y se le da el valor adecuado en la inicialización. La rutina *NextBuff* simplemente llama a esa rutina a través del puntero (si bien en Pascal se hace de forma transparente) y a continuación llama a la rutina *NextTick* para que mezcle los *buffers* del *player*.

MEZCLA ESTÉREO

Al enviar cada voz sólo al canal izquierdo o al derecho, se obtiene un resultado poco real, con los instrumentos demasiado separados, y si se intenta escuchar con unos auriculares al cabo de poco tiempo podemos terminar mareados. Para evitar este problema puede utilizarse la solución costosa de considerar tanto el volumen izquierdo como el derecho de cada voz, o bien puede hacerse una mezcla del canal izquierdo y derecho en el momento de llenar el *buffer* de DMA estéreo. Por ejemplo, se puede tomar el 75% del canal izquierdo, sumarle el 25% del canal derecho y enviar el resultado al canal izquierdo, y viceversa. Esto es lo que hacen las rutinas *CopiaBufDMA16EstéreoMix* y *CopiaBufDMA8EstéreoMix*, para *buffers* de 16 y 8 bits respectivamente. Estas rutinas son variantes de las que llenan el *buffer* de DMA estéreo, pero en lugar de simplemente mover datos, hacen algunas operaciones con ellos antes de colocarlos en el destino.

Se utilizan las proporciones 75% - 25% porque el resultado es muy agradable al oído y todavía se distingue el lado izquierdo del derecho, y sobre todo porque estas proporciones son muy fáciles de obtener con rotaciones, sumas y restas, con las que se consume poca CPU. La mezcla estéreo es opcional, así que se ha añadido un *booleano* a la estructura *TSoundMode* del fichero *SONGS.PAS*, y según el valor de este *booleano* se inicializa el puntero a la rutina de llenado mencionada anteriormente.

LA GRAVIS ULTRASOUND

Hace dos entregas se explicó cómo emular por software varios canales teniendo sólo uno (o dos en el caso de una tarjeta estéreo), a base de ir recorriendo las digitalizaciones con índices e incrementos en punto fijo, multiplicando por el volumen de cada canal y haciendo la media de cada canal para cada muestra. Este proceso de mezcla por software es el que consume más tiempo de CPU del reproductor de MODs, con

gran diferencia con respecto al resto de rutinas. Pero existe una nueva generación de tarjetas de sonido que incorporan este proceso en el hardware. Se trata de las tarjetas *WaveTable*, entre las que se encuentran la Sound Blaster AWE 32 y la Gravis UltraSound o GUS.

Las tarjetas *WaveTable* tienen su propia RAM en donde almacenan las digitalizaciones, y son capaces de reproducir varias simultáneamente: tanto la AWE 32 como la GUS pueden reproducir hasta 32 canales de 16 bits. Si bien las prestaciones técnicas de la AWE 32 superan a las de la GUS, la GUS es bastante más barata, y sobre todo tiene una ventaja decisiva para los que nos gusta programar directamente sobre el hardware: el kit de desarrollo es público y gratuito, y describe detalladamente el hardware de la tarjeta. Dicho kit puede encontrarse en cualquier BBS bajo el nombre de *GUSDK*.** y ocupa aproximadamente *1.5 MB*. Por eso, no profundizaremos demasiado en la programación de la GUS, sino que nos centraremos en cómo adaptar el reproductor para que utilice esta tarjeta.

De todas formas vale la pena hacer un rápido repaso por las características técnicas de la GUS. Como ya se ha explicado, es capaz de reproducir hasta 32 canales simultáneos de 16 bits. El "corazón" de la GUS está en el chip GF1, que se encarga de mezclar los canales muestra a muestra teniendo en cuenta todos los parámetros de cada canal. De hecho, el GF1 se comporta de forma muy similar a las rutinas de mezcla de canales por software que se explicaron hace dos entregas. Por eso, apenas tendremos que adaptar algunos parámetros y enviárselos a la tarjeta en lugar de a nuestras rutinas. La frecuencia de muestreo a la que mezcla el GF1 depende como es lógico del número de canales activos. El mínimo número de canales activos es 14, y la frecuencia correspondiente es 44100 Hz (la misma que un *compact disc*). A partir de aquí la frecuencia disminuye proporcionalmente, de forma que por ejemplo con 28 canales activos la frecuencia es la mitad: 22050 Hz. En general puede calcularse con una sencilla regla de tres:

*Frecuencia de muestreo = 14*44100 / N*

Donde N es el número de canales activos. Pero el GF1 hace más cosas que nuestras rutinas de mezcla. Por ejemplo, cada canal es panoramizable: mediante un número de 0 a 15 se puede regular por qué lado sonará, donde 0 es totalmente a la izquierda, 15 totalmente a la derecha y los demás números son posiciones intermedias (8 significa el centro).

Algo que aumenta en gran medida la calidad de la reproducción en la GUS es la interpolación. Al igual que nuestras rutinas, el GF1 recorre las digitalizaciones con índices en punto fijo. Esto significa que para frecuencias menores a la de muestreo (o sea, en la mayoría de los casos), se va repitiendo una muestra hasta tomar la siguiente. El resultado de esta repetición es una onda muy escalonada y poco natural (ver figura 2), que nuestro oído percibe como un sonido más metalizado. Para evitar este escalonamiento se interpolan los valores entre una muestra y la siguiente. En el caso de la interpolación lineal de primer orden, esto consiste simplemente en trazar un segmento entre las dos muestras, determinando de este modo los puntos intermedios. Por ejemplo, si la mezcla trabaja a 44 KHz y se reproduce una digitalización a 22 KHz, se repetirá dos veces cada muestra. Si una muestra vale 0 y la siguiente 100, en lugar de colocar 0,0,100,... en el *buffer*, colocaríamos 0,50,100,... con lo que se obtendría una onda más suave ya que se pasa de una muestra a la siguiente de forma menos brusca, resultando un sonido más natural. La GUS, además, al trabajar con muestras de 8 bits las interpola pasándolas a 16 bits, lo cual aumenta aún más la calidad del sonido.

RECUADRO 1

```
PROCEDURE GUSUpdateChannel(Chn:Byte);
VAR
  Offs:Word;
BEGIN
  GUSSetCurVoice(Chn-1);
  {Selecciona el canal a modificar}
  IF pSng^.ChnVol[Chn].Disable THEN
  BEGIN
    GUSSetVLinVol(0);
    Exit;
  END;
  pChn:=@Channel[Chn];
  WITH pChn^ DO
  BEGIN
    IF OldVol<>Volume THEN
    BEGIN
      GUSSetVLinVol(Volume);
      {Modifica el volumen según tabla}
      OldVol:=Volume;
    END;
    IF (OldPrd<>Period) AND (Period<>0) THEN
    BEGIN
      GUSSetVFreq(GusDivisor DIV Period);
      {Modifica el incremento}
      OldPrd:=Period;
    END;
    IF OldInst<>Instrum THEN
    BEGIN
      GUSSetVSamp(Instrum);
      {Modifica inicio y fin de instrumento}
      OldInst:=Instrum;
    END;
    IF Playing THEN
    BEGIN
      Playing:=FALSE;
      Offs:=PosOfs-Ofs(pSample^);
      IF Loop THEN
GUSSetVStart(Instrum,Offs,8)
      ELSE GUSSetVStart(Instrum,Offs,0);
      {Modifica posición y loop de instrumento}
    END;
  END;
END;
```

Actualización de los parámetros de un canal de la GUS.

GUS EN EL MOD PLAYER

Las rutinas de manejo de GUS están separadas en dos ficheros: *GUS.PAS* y *SONGUS.PAS*. El fichero *GUS.PAS* contiene rutinas de detección y programación directa de la tarjeta, mientras que *SONGUS.PAS* contiene rutinas que permiten que el reproductor utilice la tarjeta para reproducir MODs.

Una vez detectada e inicializada, si se utiliza una GUS para reproducir MODs lo primero que hay que hacer es enviar todas las digitalizaciones a la RAM de la tarjeta, ya que es de allí de donde tomará las muestras el GF1 para mezclarlas. Para ello se utiliza la rutina *GUSPoke*, que envía un único byte a una posición de la RAM de la GUS (una forma más eficiente pero más complicada de hacerlo sería utilizando DMA). La rutina *GUSSongSetup* se encarga de esto tras indicar el número de voces activas y de panoramizar cada canal. El volumen de cada canal de la GUS se indica con un número en coma flotante (4 bits de exponente y 8 de mantisa), y no con un byte que toma valores de 0 a 64. Por eso en la inicialización se prepara una tabla de volúmenes, en la que cada una de las 65 entradas contiene el volumen en coma flotante correspondiente.

Como ya no hay un *buffer* que se envía por DMA a la tarjeta, ya no hay una interrupción que nos avisará de que se ha terminado dicho *buffer*. Por eso se programa el conocido *timer* 0 para que genere una interrupción en cada *tick* del

MOD, es decir, 50 veces por segundo con el BPM por defecto. La GUS también tiene sus propios temporizadores que pueden generar interrupciones, pero por cuestiones de sencillez no los utilizaremos. Cuando llega una interrupción es el momento de actualizar los efectos y quizás la partitura, llamando a la rutina *NextTick*. Ahí se mira si se está haciendo una mezcla por software o si se está utilizando una GUS. En el primer caso, se hacen las necesarias llamadas a *FillBuffer* para cada canal, y en el segundo caso se llama a la rutina *GUSUpdateChannel* para cada canal, que puede verse en el recuadro 1.

Lo primero que hace *GUSUpdateChannel* es seleccionar la voz o canal que se va a modificar. A continuación modifica sólo los parámetros que han cambiado con respecto a la pasada anterior, para lo cual la estructura *TChannelInfo* contiene unos campos dedicados a memorizar el estado anterior. Ya sabemos cómo funciona el volumen: en nuestro caso se utiliza una tabla que se encarga de adaptar el volumen de MOD a volumen de la GUS. Por otra parte, la frecuencia del canal es muy fácil de calcular, ya que no por casualidad el incremento que utiliza la GUS está expresado en coma fija. Nuestras rutinas de mezcla por software utilizan un incremento con 8 bits de parte fraccionaria y 8 bits de parte entera, y el GF1 utiliza un incremento con 6 bits de parte entera y 9 bits de parte fraccionaria, pero ahí terminan todas las diferencias. Así que el incremento se calcula del mismo modo que cuando se mezcla por software: usando la siguiente regla de tres:

Inc = FrecDig/FrecDAC

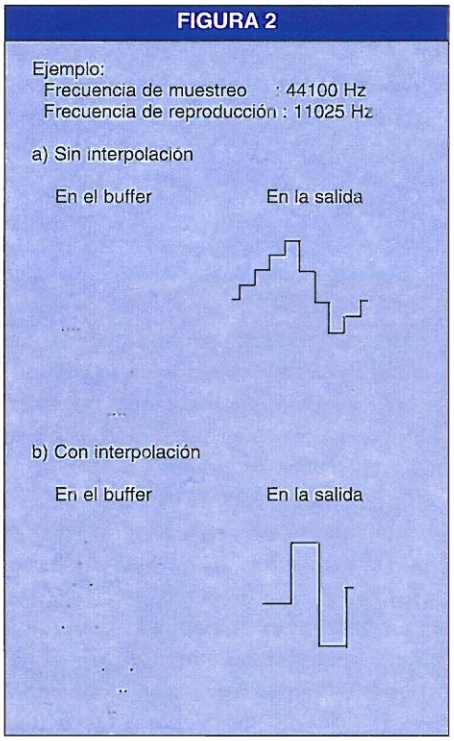
Donde *FrecDig* es la frecuencia a la que debe reproducirse el canal individual y *FrecDAC* es la frecuencia global a la que funciona la tarjeta. Ya sabemos calcular *FrecDAC* en función del número de voces activas, así que no queda más que precalcular un factor para que nos determine el incremento en función del periodo, que es lo único que se desconoce ya que varía con cada nota (ver artículo del número 11). Este factor se almacena en la variable *GUSDivisor*.

Sólo nos queda especificar el instrumento. Esto se hace modificando unos registros en los que se indica las posiciones de la memoria de la GUS del inicio, final y posición actual de la digitalización, y si el instrumento tiene *loop*, se indica alterando un bit del registro correspondiente. Para conocer todas las posiciones en la RAM de la GUS de cada instrumento, cuando se copian las digitalizaciones a la RAM de la tarjeta se memorizan las posiciones en una tabla de instrumentos, que se lee en tiempo de reproducción.

Así de sencillo es adaptar el reproductor para que utilice la GUS. Claro que las rutinas pueden mejorarse enormemente: en primer lugar, copiando los instrumentos a la GUS vía DMA, lo cual es mucho más rápido. De hecho, al cargar un MOD utilizando la GUS puede dar la impresión de que el PC se ha colgado, cuando en realidad se está tomando su tiempo para subir muestra a muestra las digitalizaciones a la tarjeta. También sería conveniente utilizar los *timers* e interrupciones de la GUS, lo cual dejaría al *timer 0* e *IRQ 0* libres para otros menesteres y haría nuestras rutinas más genéricas y versátiles. Y hay un tema que no se ha comentado: los famosos *UltraClicks*.

Como la GUS interpola valores entre muestra y muestra, la calidad resultante es mucho mayor que con una mezcla por software convencional. Pero paradójicamente, al tenerse mayor calidad, los cambios bruscos en un canal (cambios de instrumento, de volumen, *loops* mal enlazados, final de instrumento, etc.), provocan un salto anormal en la salida que nuestro oído percibe como un *click*. Estos *clicks* pueden percibirse con el programa que acompaña este artículo, según el MOD que se escuche. Para evitar parte de los *clicks* basta con copiar la última muestra detrás de un *loop* o de un final de instrumento, pero los cambios bruscos de instrumento no se pueden evitar en tiempo de carga. Para evitarlos se utiliza la técnica de las rampas de volumen: debe hacerse una rápida rampa de volumen hacia el 0 al terminarse un instrumento y una rápida rampa hacia el volumen total al empezar uno nuevo. Pero esto ya es más complicado y las rutinas que se suministran no lo incorporan.

FIGURA 2



Interpolación lineal.

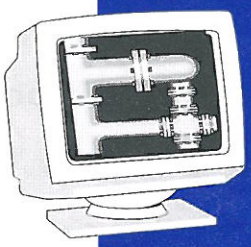
EN LA PRÓXIMA ENTREGA

Nuestro reproductor (o *player* si se prefiere en inglés), cada vez está más completo, pero aún está lejos de estar acabado. Faltan por implementar algunos efectos menos frecuentes, y otros no tan extraños como son el cambio de BPM. También falta por implementar un control de saturación, que permita utilizar volúmenes superiores a los permitidos en función del número de canales, y podría tenerse en cuenta el volumen izquierdo y derecho de cada canal y no sólo el global. Podría añadirse interpolación por software, a pesar de que consume mucha CPU ya que se requiere una multiplicación adicional por cada muestra. Sería interesante cargar otro tipo de ficheros como por ejemplo los S3M, que son unos ficheros con un formato similar al MOD que crea el programa Scream Tracker III. Y las rutinas de mezcla no están nada optimizadas: para empezar podría utilizarse una tabla en lugar de multiplicar por el volumen para cada muestra.

Como puede verse, el tema todavía da mucho de sí. Pero para no encasillarnos demasiado en el tema MOD, también se explicará cómo programar la tarjeta de sonido para digitalizar, es decir, para leer datos del ADC y recogerlos en la memoria del PC.

CTHREADS THREADS PARA MS-DOS

Enrique Velasco



Muchos sistemas operativos ofrecen la posibilidad de crear tareas, esto es, de hacer que la CPU ejecute de modo aparentemente simultáneo varios programas. En realidad, lo que sucede, al menos en máquinas con un sólo procesador, es que se reparte el tiempo de la CPU de manera que parece ejecutar varios programas al mismo tiempo, ejecutándose en un cierto orden secciones de código de cada uno de ellos.

La elección del momento en que se produce el salto de una tarea a la siguiente permite hacer una primera clasificación: por un lado tenemos la multitarea *apropiativa* (*preemptive*), característica de Unix , OS/2 y Windows NT, y por otro la multitarea cooperativa, propia de Windows 3.X o NetWare. En el primer caso, la conmutación de tareas se produce de manera forzada, mediante una interrupción, mientras que en el segundo caso la tarea cede el control voluntariamente al llamar a servicios del sistema operativo, tales como *GetMessage()*, de Windows.

La multitarea apropiativa tiene como principal ventaja garantizar un reparto equitativo del tiempo de CPU entre tareas con la misma prioridad. La multitarea cooperativa, por su parte, exige una programación más cuidadosa, para evitar que una misma tarea se quede con la CPU durante un tiempo excesivo, pero proporciona en contrapartida una eficiencia máxima, al reducir el número de cambios de tarea al mínimo imprescindible.

En sistemas operativos clásicos, en particular en Unix, cada tarea tiene un espacio de direcciones lógicas separado y protegido. Si se quiere compartir memoria entre dos o más tareas, hay que llamar a servicios específicos del sistema. A este tipo de tarea se le aplica el calificativo de *heavyweight* (peso pesado). Un inconveniente no despreciable de las tareas *heavyweight* es la cantidad de información de contexto que hace falta salvar y restaurar durante la conmutación de una tarea a otra.

Cuando la multitarea funciona en el interior de un único programa, las características de protección de memoria pierden importancia, y por contra se hacen más gravosos los cambios de contexto.

El estado de ejecución de un thread está caracterizado por los registros de la CPU y la pila

De la conveniencia de soslayar estas desventajas surge el concepto de *thread*. Los *threads*, también llamadas *lightweight tasks* son como tareas dentro de un solo programa, pero comparten el mismo espacio de direcciones lógicas, lo que ahorra mucho tiempo en las conmutaciones de *threads*.

Windows NT, Windows 95, OS/2 y muchas versiones modernas de Unix permiten que cada tarea *heavyweight* se pueda descomponer a su vez en varios *threads* con flujos de control independientes.

Con la pequeña librería presentada en este artículo se puede experimentar con las posibilidades de programación que ofrecen los threads sin cambiar de sistema operativo y con un compilador "clásico" de 16 bits como es Turbo C.

El conjunto de funciones que presentamos en este artículo permite manejar *threads* dentro de un sistema operativo esencialmente monotarea, como es MS-DOS.

CTHREADS

Es un hecho bien conocido que MS-DOS no es reentrante, es decir, no permite llamar a uno de sus servicios hasta que haya terminado de ejecutarse cualquier llamada anterior. Ésta es una de las características que más quebraderos de cabeza crean a los que escriben *drivers* o programas residentes, y también tiene gran importancia a la hora de implementar la multitarea.

El motivo de esta situación es que una de las primeras cosas que hace DOS al ejecutar una llamada es poner los registros SS y SP apuntando a una zona de *stack* propia. Esta medida no es arbitraria, ya que permite que las aplicaciones llamen a DOS sin tener que tomar ninguna precaución especial respecto al tamaño de *stack* necesario.

Si se trabaja con multitarea apropiativa, puede suceder que mientras se ejecuta una llamada a DOS se le retire el control a la tarea actual y se le de a otra que a su vez llame a DOS. En este caso, lo que hubiera en el *stack* de DOS en el momento de la interrupción, incluyendo la dirección de retorno, queda sobrescrito por la segunda llamada, lo que produce con casi total seguridad un bloqueo de la máquina.

Hay dos posibles soluciones a esta situación. La primera consiste en proteger absolutamente todos los puntos de entrada de DOS y BIOS mediante mecanismos de exclusión mutua, como semáforos o *spinlocks*. La segunda, mucho más sencilla y eficiente, consiste en emplear el enfoque cooperativo, que es lo que hemos hecho en nuestro paquete de *threads*.

Cualquier programador está familiarizado con la idea de que un mismo trozo de código actúe sobre diferentes conjuntos de datos, por ejemplo en llamadas consecutivas a una misma función. En este caso, se crea en el *stack* una estructura (denominada usualmente *stack frame*) que contiene tanto los parámetros como los datos locales de la función. La función se ejecuta con continuidad de principio a fin, y al terminar se libera el espacio de *stack* anteriormente ocupado, el cual puede ser reutilizado para cualquier otra llamada a función sin peligro de interferencia entre las llamadas.

El concepto de *thread* es más genérico. Un *thread* no tiene por qué tener un principio o un fin determinados, sino que tiene su propio flujo de control que va recorriendo el código en paralelo con otros *threads*. De este modo, varios *threads* pueden estar, por ejemplo, compartiendo código de una misma función sin tener que esperar cada una a que las otras terminen de ejecutar sus llamadas a esa función, siempre que ésta sea reentrante (posteriormente veremos cómo tiene que estar escrita la función para ser reentrante).

Existen dos tipos de multitarea, la apropiativa y la cooperativa

Si bien el concepto de *thread* no coincide con el de función, habitualmente, y por comodidad, se encapsula el código de una *thread* dentro de una función C. Esto permite emplear directamente el nombre de la función como puntero al comienzo del código del *thread*.

En cualquier momento, y a cualquier profundidad dentro de una cadena le llamadas, un *thread* puede decidir ceder el control al siguiente. Para que el que cede el control pueda continuar su ejecución exactamente en el punto en que lo dejó es preciso guardar toda la información que define su estado en ese momento.

El estado de ejecución de un *thread* en un instante dado queda completamente caracterizado por el contenido de los registros de la CPU y el estado de su segmento de *stack*. Cada *thread* en ejecución debe tener su propia zona de *stack*, ya que en caso contrario unos *threads* sobrescribirían los datos locales de otros, creando un problema análogo al que antes hemos descrito para MS-DOS.

Para conseguir esto, antes de ceder el control a un *thread* por primera vez, se reserva con malloc() una zona de memoria para su *stack*, y se colocan los registros SS y SP apuntando a esa zona.

Si bien los datos locales (dinámicos) de un *thread* y de todas las funciones que esta llama son propiedad privada de la misma, al crearse cada *stack frame* en la zona de *stack* correspondiente, no sucede lo mismo con los datos globales y estáticos. Los datos globales, como cabría esperar, son accesibles por todos los *threads*. Los datos estáticos, definidos dentro de la función que contiene el código del *thread*, son comunes a todos los *threads* que se hayan creado compartiendo ese mismo código, pero no son accesibles al resto de los *threads*. Esta última propiedad se debe a que para este tipo de variables el compilador produce, directamente en el código compartido, referencias absolutas al segmento de datos.

Por esta causa, si se desea que una función sea reentrante, es decir, que se pueda ceder el control en cualquier punto de su interior, es necesario que no acceda a variables globales o estáticas, de manera que en cualquier momento sólo pueda modificar los datos privados del *thread* que tiene el control.

Una vez que el *thread* ha comenzado su ejecución, se emplean las funciones estándar setjmp() y longjmp() para guardar y restaurar el contenido de los registros de la máquina. Dado que SS y SP forman parte del conjunto de registros manejado, se salva y restaura también de este modo la situación del *stack* de cada *thread*.

Así pues, cuando un *thread* cede el control se ejecuta la siguiente secuencia de acciones:

- se guarda el estado del *thread* actual mediante setjmp()
- se selecciona el siguiente *thread* a ejecutar
- se carga su estado y se le da control con longjmp()

La acción de decidir cuál es el siguiente *thread* que va a recibir control se llama *planificación* (en muchas ocasiones se emplea la palabra equivalente en inglés, *scheduling*).

Existen muchas posibles políticas de planificación: con o sin prioridades, con prioridades fijas o variables, etc. Para CTHREADS, la política de planificación elegida es una de las más sencillas posibles, la denominada *round-robin*. Todos los *threads* tienen la misma prioridad, y cuando un *thread* determinado cede el control, se coloca en una cola de *threads* preparados para ejecutar, cola que se gestiona en modo FCFS (*First Come First Served*), de manera que el *thread* que lleva más tiempo listo es el que primero recibe control.

Para coordinar o sincronizar la ejecución de varios *threads* puede ser necesario hacer que un *thread* se quede "suspendido", esto es, que no pueda reanudar su ejecución hasta que sea explícitamente reactivado. En CTHREADS hay dos servicios que sirven para suspender y reactivar *threads*, y que dejan la puerta abierta para la construcción de servicios de sincronización más complicados, tales como semáforos, monitores o regiones críticas.

DESCRIPCIÓN DETALLADA DE LOS SERVICIOS

void cthreads_init(void);

Esta función lleva a cabo la inicialización de las estructuras de datos internas de la librería. Debe ser llamada antes que ninguna otra.

*int cthreads_create(void *codigo, unsigned tamano_stack);*

Crea un *thread* y prepara su ejecución. El parámetro *código* debe contener la dirección de comienzo del código del *thread*; normalmente se pasará en este parámetro el nombre de una función C . El otro parámetro contendrá el tamaño de la zona de *stack* que se quiere reservar para el *thread*. En caso de que no haya memoria suficiente o espacio disponible en las tablas, la función devolverá el valor (negativo) THREADS_NOMEM, definido en cthreads.h . Si no hay problemas, el valor devuelto es un número mayor o igual a cero que se utiliza como identificador único para cada *thread*.

int cthreads_getid(void);

Esta función devuelve el identificador del *thread* que le llama.

- *void cthreads_bye(void);*

Acaba definitivamente la ejecución del *thread* que hace la llamada y deja libre toda la memoria que tuviera asignada.

int cthreads_first(void);

Este servicio inicializa el buffer de entorno de cada *thread*, y debe ser la primera función llamada dentro del código del *thread*. La llamada se debe hacer del siguiente modo:

```
if(cthreads_first())
        {
        return;
        }
```

void cthreads_next(void);

Este servicio cede el control a el siguiente *thread* listo para ejecutarse, y pone en la cola al que hace la llamada.

void cthreads_suspend(void);

Este servicio cede el control al siguiente *thread* listo, dejando suspendida al que hace la llamada hasta que se invoque el servicio *cthreads_resume()*.

void cthreads_resume(int id);

Pone en la cola de *threads* listos a aquel cuyo identificador se le pasa como parámetro, siempre y cuando su ejecución estuviera suspendida.

void cthreads_start(void);

Esta llamada se ejecuta una sola vez, y pone en marcha todos los *threads* que se hayan creado hasta el momento. Los *threads* que se creen posteriormente comenzarán su ejecución tan pronto como les toque el turno.

void cthreads_end(void);

Acaba la ejecución de todos los *threads*.

BUSCAMOS A LOS MEJORES

- Programadores en C/C++
- Grafistas, diseñadores, dibujantes
- Expertos en lenguaje ensamblador
- Programadores 3D
- Expertos en Sonido
- Músicos con nociones de MOD, MIDI
- Programadores de juegos
- Animadores gráficos
- Infografistas con experiencia en 3D Studio, Photoshop, Deluxe Paint Animation
- Programadores con dominio de lenguajes multimedia: Authorware, Toolbook, Visual Basic, Macromedia Director, etc
- Expertos en comunicaciones

PARA DESARROLLAR

SOFTWARE MULTIMEDIA:

Presentaciones, libros interactivos, programas educativos, sistemas de comunicaciones, centros servidores de datos, videojuegos y mucho más...

SI ERES programador, músico o infografista y tienes ideas o proyectos en estudio de desarrollo ponte ya en contacto con nosotros.

TE OFRECEMOS las mejores condiciones y apoyo para producir tus programas y venderlos en el mercado nacional y extranjero. Formación en nuevas tecnologías y la mejor biblioteca de rutinas gráficas y sonido para desarrolladores.

Aportamos gráficos, rutinas o música, según las necesidades, para complementar cada proyecto.

Si estás interesado en unirte a una de las empresas más punteras en alta tecnología y desarrollo de software, no dejes pasar esta oportunidad. Envíanos carta con tus datos personales (curriculum vitae, con una muestra de tus anteriores trabajos) y un teléfono de contacto a:

DDM DIGITAL DREAMS MULTIMEDIA

DIGITAL DREAMS MULTIMEDIA
Ref. Programadores
C/ Vicente Muzas 15, 1º D - 28043 MADRID
Tf.: (91) 519.23.53 Fax (91) 413.55.77
BBS: (91) 519.75.75 Internet: ddm@servicom.es

CÓMO CALCULAR PUNTOS 3D

Enrique de Alarcón

Los ordenadores de hoy en día nos ponen a nuestra disposición una gran potencia de cálculo gracias a la cuál , podemos realizar simulaciones virtuales 3d de gran calidad gráfica. En éste artículo explicamos los métodos matemáticos y gráficos para realizar éstos mundos virtuales.

A medida que evolucionan los ordenadores, la capacidad de cálculo de éstos se va multiplicando y con ello las posibilidades que ponen a disposición del usuario , el cuál a su vez, exige cada vez más de las aplicaciones que usa.
Una de las más atractivas aplicaciones de ésta potencia matemática de cara al público en general, es la representación de gráficos tridimensionales ya sea para crear presentaciones, juegos, o simuladores de realidad virtual, los cuales son el futuro del mundo del entretenimiento y de la investigación.
Los métodos de cálculo y dibujado usados para la creación de las imágenes 3D son muchos y variados.

En éste artículo explicaremos el método matemático usado para el cálculo de puntos 3D en el espacio y que se usa en todos los simuladores virtuales basados en formas poligonales. A si mismo, explicaremos también el método para calcular los polígonos visibles y no visibles que habrá en las figuras 3D que hayamos creado en base a éste sistema.

COMO SE CALCULA UN OBJETO 3D

Cuando por ejemplo vemos una nave espacial en un simulador de naves, lo que estamos viendo es un conjunto de puntos 3D (3 coordenadas X,Y,Z por punto) los cuáles se usan como vértices de referencia para los polígonos de la superficie de los objetos que vemos en pantalla. Por ejemplo, para poder representar un cubo en medio de una habitación virtual, debemos tener las coordenadas X,Y,Z de los 8 vértices que lo forman y aplicarle a cada uno de ellos todas las fórmulas de cálculo, que calcularán respectivamente, su situación angular, su posición en el espacio virtual y su proyección en pantalla.

TIPOS DE SIMULACIONES VIRTUALES

Según la aplicación que queramos dar a la simulación y de como queramos ver y manipular los objetos 3D que la compongan, usaremos uno de los siguientes tipos de simulación virtual: *Objetiva*, *subjetiva* o una *mixta*.

Simulación objetiva:
Cuando queramos ver por ejemplo un objeto en el centro con la posibilidad de aplicarle rotación, traslación y caída sin que cambie nuestro punto de vista, nos basaremos en una simulación objetiva, que sería como coger un objeto con la mano y darle vueltas frente a nosotros.
Simulación subjetiva:

Los métodos de cálculo y dibujado usados en la creación de las imágenes 3D son muchos y variados

Es aquella en la que el sujeto está inmerso en el entorno 3D y el cambio de alguno de los 3 ángulos equivale a mover la cabeza dentro de ese entorno. Por ejemplo, podríamos estar caminando por el interior de un castillo y el cambiar el ángulo vertical (caída) cambiaría el ángulo vertical de nuestra cabeza (todo parecería moverse menos nosotros).

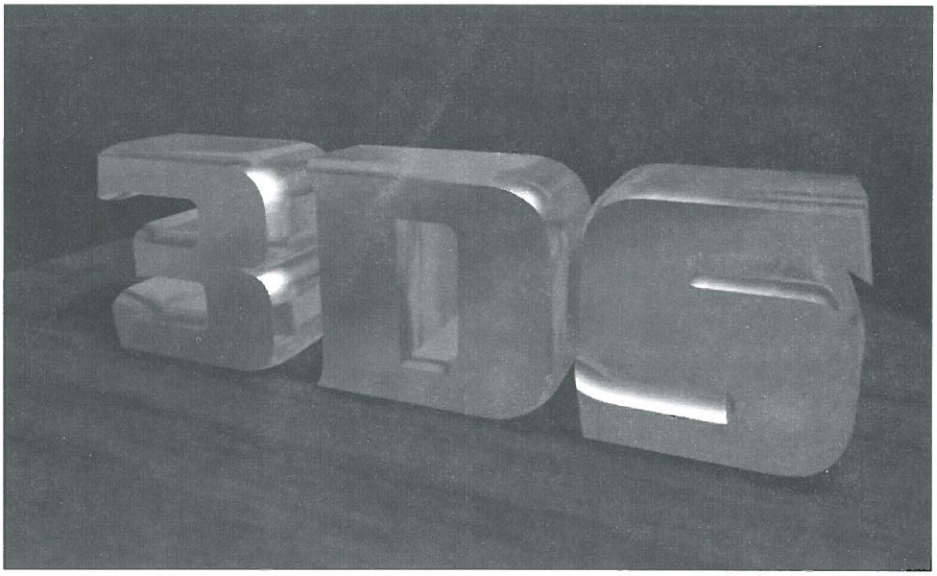
La única diferencia que hay entre los dos tipos de mundos virtuales está en el orden de los cálculos en las fórmulas matemáticas que aplicamos a los puntos 3D usados como vértices de los objetos.

Simulación mixta:
Usaremos un sistema mixto cuando queramos ver por ejemplo, un avión realizando un giro de 360 grados respecto a un terreno fijo. En éste caso debemos primero debemos calcular los vértices del avión de forma aislada como si fuésemos a verlo en un entorno objetivo y almacenar los valores 3D sin calcular la proyección 2D. Una vez hecho esto cogemos éstos nuevos puntos 3D y les aplicamos a su vez los cálculos como si fuesen los vértices iniciales de un objeto normal 3D en un entorno objetivo.

PASOS A REALIZAR PARA CALCULAR UN PUNTO 3D

Primero conozcamos todas las variables que tenemos que saber para los cálculos:

1º Las coordenadas X,Y,Z del punto en el espacio (X,Y,Z).
2º Las coordenadas del sujeto observador (donde está el observador Ox,Oy,Oz).
3º Valores angulares de los 3 tipos de giros que podemos aplicar al punto, que son rotación, balanza y caída (Ar1,Ar2,Ar3) que se darán en radianes (628 grados son 6.28 radianes):
4º Factor de distorsión angular, que es la deformación de la imagen que se produce en pantalla debido a la proximidad del punto. (A más proximidad más se deforma la imagen que vemos). En casos normales lo dejaremos con el valor 1200.



Esquema con los 3 tipos de giros que podemos aplicar a los objetos 3D.

Para obtener el punto 3D en la pantalla 2D hay que calcular 3 cosas:

1º Aplicar los 3 giros angulares al punto mediante cálculos trigonométricos.
2º Trasladar el punto por el espacio virtual.
3º Calcular la proyección del punto 3D resultante en la pantalla 2D.

Las fórmulas de cada paso quedarían como sigue:
Fórmulas de rotación:
XA = COS(Ar1)*X - SEN(Ar1)*Z
ZA= SEN(Ar1)*X + COS(Ar1)*Z
fX= COS(Ar2)*XA + SEN(Ar2)*XA
YA= COS(Ar2)*Y - SEN(Ar2)*XA
fZ= COS(Ar3)*ZA - SEN(Ar3)*YA
fY= SEN(Ar3)*ZA + COS(Ar3)*YA
Fórmulas de traslación:
fX=fX+MX
fY=fY+MY
fZ=fZ+MZ
Fórmulas de proyección:
SX= D*fX/fZ
SY= D*fY/fZ

El procedimiento explicado se usa en las simulaciones objetivas, en caso de querer calcular la versión subjetiva, sólo tenemos que cambiar el orden de los 3 cálculos. Primero calcular la traslación, segundo la rotación y por último la proyección.

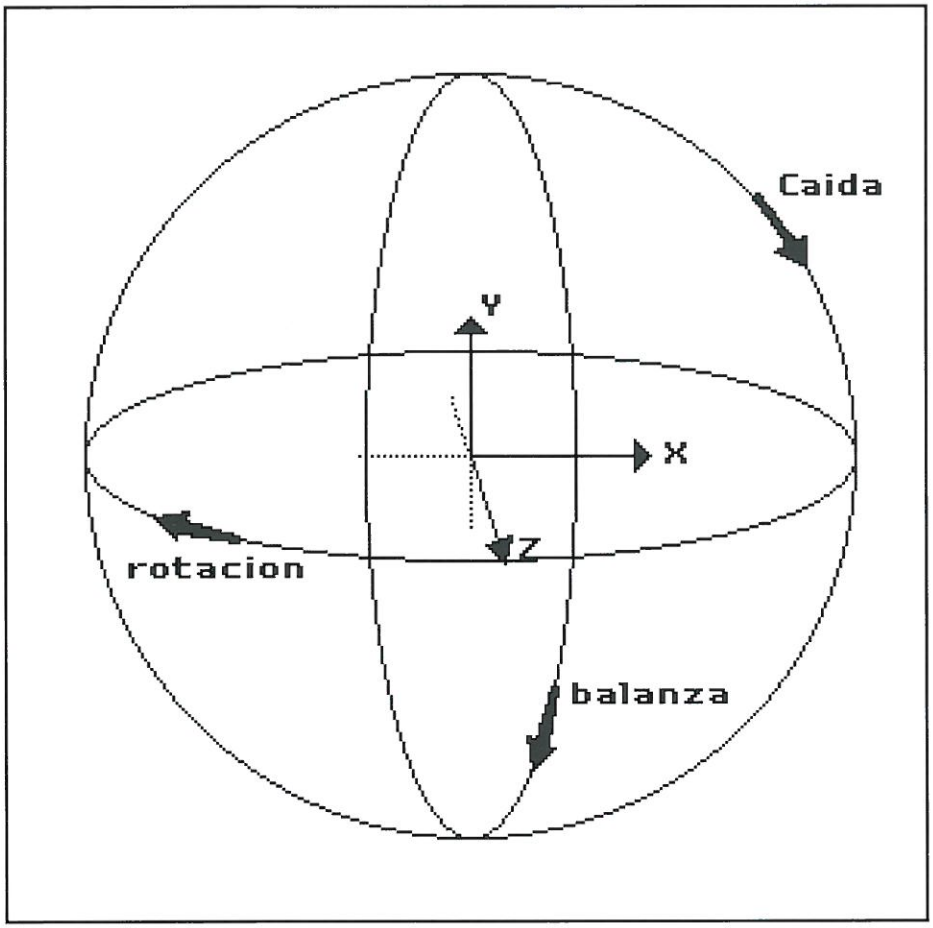
EJEMPLOS DE APLICACIÓN

Realizaremos el ejemplo más sencillo que se nos puede plantear. Dibujar en pantalla (o sea calcular) los puntos usados como vértices de un cubo en la pantalla 2D de nuestro monitor.
Un cubo tiene 8 vértices, por lo tanto , tendrá ocho puntos con 3 coordenadas por punto , que serán:
A(10,-10,10) , B(10,-10,-10) , C(-10,-10,-10) , D(-10,-10,10) , E(10,10,10) , F(10,10,-10) , G(-30,-30,-30) , H(-30,30,30).

Las coordenadas donde está situado el observador serán por ejemplo Ox=0 , Oy=0 , Oz=-400 y los valores de giro a aplicar al objeto serán: Rotación Ar1=5-5 , balanza Ar2=0.0 , caída Ar3=4.4.
Dando a cada variable global su valor, sólo tenemos que realizar los 3 cálculos con cada punto 3D cambiando cada vez los valores de entrada. Los valores resultantes son coordenadas 2D de pantalla.

Si queremos por ejemplo, ver el cubo como forma *alámbrica* y no como puntos, sólo tenemos que guardar las ocho coordenadas 2D de pantalla

En simulaciones subjetivas sólo cambiamos el orden de las fórmulas 3D



Ejemplo de imagen generada por *RAYCASTING*.

obtenidas y dibujar líneas que unan todos los vértices.

Si queremos ver un cubo sólido, sólo tenemos que dibujar todos los polígonos que estén visibles (de los 6 totales que tiene) uniendo los 4 vértices correspondientes de cada cara (los cuatro puntos de cada lado de su superficie).

ELIMINACION DE LAS CARAS OCULTAS DEL CUBO

Si queremos poder representar el cubo en forma poligonal sólida (para seguir el ejemplo), debemos saber cuáles son las caras del cubo que se ven y las que no, ya que de lo contrario se pueden imaginar el resultado. Para calcularlo, utilizaremos el llamado método de la ecuación del plano.

Para calcular si una cara es visible al observador, necesitamos los valores X,Y,Z obtenidos antes de la proyección de al menos tres puntos consecutivos de el polígono en cuestión y entrarlos como parámetros en la siguiente fórmula de forma antihoraria (primero el tercer punto y último el primero):

Ecuación del plano:

VP1=fX1* (fY2*fZ3 - fY3*fZ2)
VP1=-1*VP1
VP2=fX2*(Y3*fZ1 - fY1*fZ3)
VP3=fX3*(Y1*fZ2 - fY2*fZ1)
VISIBILIDAD = VP1 - VP2 - VP3

Tras calcular éstas fórmulas, obtendremos un número en coma flotante que indicará la visibilidad de la superficie. Si *VISIBILIDAD* es menor que 0 será visible y por lo tanto se debe dibujar, si *VISIBILIDAD* es mayor o igual a 0, la superficie no debe dibujarse.

Los cálculos 3D en una simulación objetiva son rotación, traslación y proyección

LIMITACIONES DE LA ECUACIÓN DEL PLANO

Este método para calcular las superficies visibles, nos será útil en los casos en que el objeto a dibujar sea simple y siempre y cuando las caras no formen zonas convexas que se "metan hacia dentro" del objeto (como por ejemplo una pelota "chafada" hacia dentro). En caso de que sea un objeto complejo, se debe recurrir a métodos de cálculo más complejos como el *Z-Buffer*.

EL MÉTODO Z-BUFFER

El método *Z-buffer* consiste en tener un duplicado en memoria con 16 o mas bits por cada pixel de pantalla. Cuando dibujamos cada polígono, debemos ir rellenando en el segundo buffer la distancia Z a la que está cada pixel que vamos dibujando en pantalla, lo cuál podemos deducirlo a partir de las distancias Z que hemos obtenido al calcular los vértices de los polígonos. Después, cada vez que dibujamos un nuevo punto en pantalla, sólo debemos calcular si el nuevo pixel ésta más cerca que el que ya tenemos dibujado (en caso de que ya haya alguno), si está más cerca, lo dibujamos, si no, lo saltaremos.

Este método , aunque resulta mucho más lento y pese a que necesita bastante memoria para funcionar (más aun si estamos en un modo SVGA) es siempre infalible , y no importa la complejidad de los objetos a dibujar , puesto que siempre saldrá una imagen perfecta.

OTROS SISTEMAS

Otro sistema que tiene mucha aceptación entre los programas de *render*, es el método de trazado de rayos o llamado también *RAYCASTING*, el cual se basa en proyectar un haz de luz por cada pixel y calcular en el lugar del escenario donde rebota.

Este método, aunque es muy lento y complejo de cálculo, permite realizar efectos especiales de trasparencia, deformación de lentes, focos de luz y todo lo que nos podamos imagi-

SPI

SOFTWARE PRODUCTS INTERNATIONAL

SEDYCO

Mayorista - Distribuidor Exclusivo,
Península Ibérica y Países Iberoamericanos

AHORRE DINERO

WindowBase 2.0

EL SISTEMA DE GESTIÓN DE BASES DE DATOS RELACIONALES, BAJO WINDOWS, BASADO EN EL MOTOR SOFTWARE DEL MÍTICO PAQUETE INTEGRADO "OPEN ACCESS"

OFERTA* DE LANZAMIENTO

¡24.990 pts.!

(IVA y Portes, no incluidos)

SUPER-OFERTA COMPLEMENTARIA
(restringida a lectores de "SÓLO PROGRAMADORES")

¡24.900 Ptas.! (I.V.A. y portes no incluidos)

Compuesta de:

1. Paquete integrado OPEN ACCESS IV, con entorno programador (RUN TIME, opcional)
2. Manuales eletrónicos (2.000 páginas), para acceso y consulta por pantalla.
3. Curso multimedia MECA-VOICE, de mecanografía por ordenador.

WindowBase 2.0
Copyright 1987 - 1995 Software Products International
Traducción y adaptación de la versión española, en castellano
SEDYCO - Servicios, Desarrollos y Consulting, S.A.
Colaboración con SEDYCO, S.A. (Contratos de 15 Junio 1995),
para la REVISION TECNICA Y LINGUISTICA DE LA TRADUCCION:
UNIV. DE ALCALA DE HENARES
Grupo de profesores del Departamento de Automática
UNIV.POLITÉCNICA DE MADRID
Grupo de profesores del Departamento de C.E.I.

Pantalla real, de arranque e iniciación

UN DECÁLOGO DE DIFERENCIAS, SIN RIVAL

Ventajas Técnicas Competitivas

1. Potente Lenguaje de Programación de Aplicaciones (**DASL**: Data Access Script Language)
2. Drivers de conectividad con otros SGBDR (**XBASE, ODBC**, etc.)
3. Compatibilidad absoluta con el paquete integrado **OPEN ACCESS** y otros **estándares del mercado**

Ventajas Operativas Cualitativas

4. Manuales y Software del Producto, traducidos en España, con la colaboración de la **Universidad de Alcalá de Henares** y de la **Politécnica de Madrid**
5. **Instalación indistintamente** del Software del Producto, tanto en configuración monousuario como en red local

Ventajas de Calidad de Servicio

6. **Servicio Técnico Oficial** propio, **y de Consultoría** a nivel nacional, con software del fabricante y asesoría puntual de Departamentos universitarios
7. **Revista Técnica trimestral**, monográfica y exclusiva para los usuarios de **WindowBASE** y **OPEN ACCESS**
8. **Club de Usuarios "PUERTAS ABIERTAS"**, con ventajas específicas de servicios, informaciones y ofertas económicas

Ventajas de Atención a la Cualificación

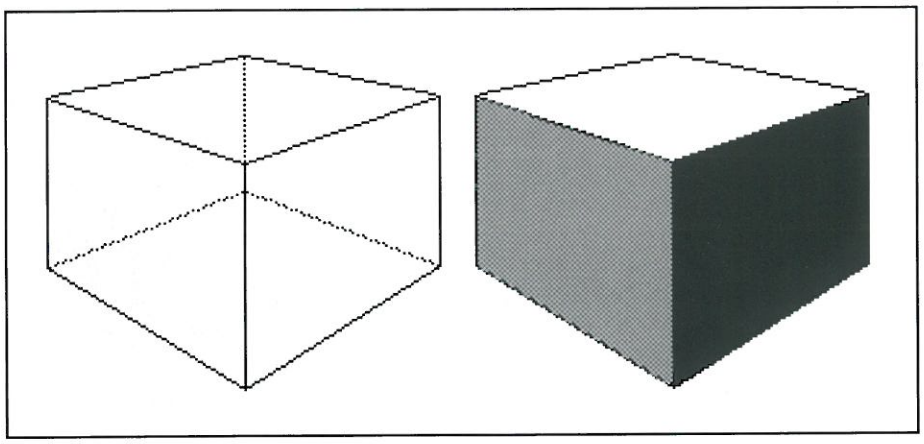
9. **Programas específicos y sistemáticos de Formación**, con la Universidad, a nivel usuario y de técnicos en programación
10. **Plan Nacional de Exámenes** de Técnicos, Especialistas y Profesores Autorizados, con **Titulación Oficial de SOFTWARE PRODUCTS INTERNATIONAL**

***LLAME A SEDYCO, al Tel.: (91) 311 36 19 o al FAX: (91) 450 04 24, Y LE INFORMAREMOS**

(También puede utilizar el teléfono exclusivo de ventas: 908 80 00 33)

LECTORES DE "SÓLO PROGRAMADORES"

DESCUENTO ADICIONAL ESPECIAL DEL 15%
(No acumulable a otras promociones)



Ejemplo de un cubo 3D en representación *alámbrica* y *sólida*.

nar. Esta técnica es la usada por programas profesionales como son *3D Studio* y similares.

Juegos famosos como el Wolfenstein 3D , el DOOM y otros similares, se basan en técnicas optimizadas y reducidas del *RAYCASTING* para reproducir escenas fotorrealistas de alta velocidad.

ACELERACIÓN DE LAS FÓRMULAS 3D

Los cálculos para los puntos 3D se hacen tediosos en caso de que el número de vértices sea elevado. Si queremos realizar por ejemplo un videojuego y necesitamos minimizar el tiempo de cálculo para poder realizar mayor número de frames por segundo, podemos limitar por ejemplo, el número de giros posibles, ya que por ejemplo, en juegos tipo *Wolfenstein* sólo necesitamos calcular la rotación, caída y balanza no se usan y por lo tanto, podemos simplificar las fórmulas ya que los cosenos *COS(Ar2)* y *COS(Ar3)* quedarán con el valor constante de 1 y los senos *SEN(Ar2)* y *SEN(Ar3)* quedarán con el valor constante 0. Las fórmulas que en éste caso se simplificarían como sigue:

Fórmulas de rotación:

XA = COS(Ar1)*X - SEN(Ar1)*Z
ZA= SEN(Ar1)*X + COS(Ar1)*Z
fX= 1*XA + 0*XA
YA= 1*Y - 0*XA
fZ= 1*ZA - 0*YA
fY= 0*ZA + 1*YA

Es resultado con las constantes sería:

XA = COS(Ar1)*X - SEN(Ar1)*Z
ZA= SEN(Ar1)*X + COS(Ar1)*Z
fX= XA
YA=Y
fZ= ZA
fY= YA

La simplificación final sería:

fX = COS(Ar1)*X - SEN(Ar1)*Z
fZ= SEN(Ar1)*X + COS(Ar1)*Z
fY= Y.

Como se puede comprobar , el ahorro de cálculos cuando podemos dejar *Ar2* y *Ar3* sin alterar es muy grande. Y si quisiéramos dejarlo al mínimo , podríamos crear dos tablas precalculadas con todos lo senos y cosenos precalculados con los incrementos que puede usar el programa, pues las instrucciones de cálculo trigonométrico son de las más lentas del procesador.

También, si las escenas contienen muchos objetos virtuales, podemos hacer que sólo se calculen para representar en pantalla los objetos que estén a una distancia Z mínima que considere el programador. En caso por ejemplo de un simulador de vuelo donde hubiesen 30 aviones en movimiento, sólo sería necesario calcular los aviones que estuviesen a menos de 10 kilómetros virtuales, de los cuales, el programador es el que debe poder deducir a que Z máxima equivalen.

Las caras ocultas de un objeto simple podemos eliminarlas con la llamada <Ecuación del plano>

Limitar el rango de giros posibles puede cuadruplicar la velocidad de calculo

RESUMEN

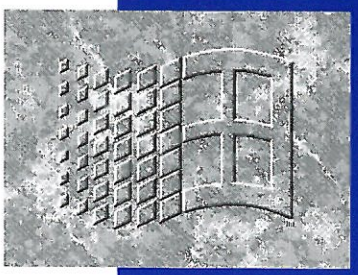
Primero hemos explicado en que se basan los objetos tridimensionales que vemos en las simulaciones virtuales 3D típicas. Hemos visto también, que hay tres maneras de plantear la representación por pantalla, dependiendo del objetivo de la representación, podemos considerar tres tipos de simulaciones virtuales, las objetivas, las subjetivas y las mixtas, que no son más que una mezcla de los dos anteriores. Tras ver los tipos de entornos , hemos dado las fórmulas necesarias para calcular un punto 3D en el espacio en una simulación objetiva y hemos explicado los cambios que se dan en las subjetivas. Para clarificar, hemos dado un ejemplo básico de representación de un cubo en el espacio y hemos explicado también como se dibujan sus superficies a base de polígonos. Para acabar, hemos dado el método más sencillo usado para la eliminación de las caras ocultas de un objeto 3D en pantalla (superficies no visibles desde el observador) y también hemos dado datos orientativos de otros métodos más complejos usados en aplicaciones más profesionales.

BIBLIOGRAFÍA

Programación gráfica en C , Técnicas avanzadas de modelado acabado y animación en 3D. Lee Adams. Anaya Multimedia.

MÉTODOS DE E/S. FICHEROS DE INICIALIZACIÓN

Jorge R. Regidor



En la entrega anterior de nuestro editor particular, se dejó pendiente la implementación de la funcionalidad de configuración, así como no se entró en detalle sobre los métodos de acceso a ficheros, los cuales ya se utilizaron para leer/escribir sobre los ficheros de texto. En esta entrega se va a hablar sobre los métodos de acceso a ficheros, así como de la creación y mantenimiento de los ficheros de inicialización. Para esto, se creará un fichero de inicialización (WINEDIT.INI), donde se guardarán los datos necesarios para la correcta carga del programa con las opciones elegidas por el usuario.

MÉTODOS DE E/S DE FICHEROS

Como se ha indicado con anterioridad, los métodos de E/S de ficheros son quizá la parte de la programación para Windows que más se parece a la programación para DOS. Sin embargo, tiene sus diferencias, las cuales vamos a intentar resaltar sobre el resto. Para detallar este apartado se va a hablar primero de las funciones de apertura de ficheros, luego de la de cierre y posteriormente de las funciones de lectura/escritura de ficheros.

APERTURA DE FICHEROS

Las funciones encargadas de la apertura de un fichero son dos: *OpenFile* y *_lopen*. Ambas tienen la misma misión, pero, sin embargo, *OpenFile* es más potente, permitiéndonos disponer de mayor información sobre el fichero abierto o mostrar automáticamente un diálogo de error si un fichero no ha sido encontrado. La función *OpenFile* tiene el siguiente formato:

```
HFILE OpenFile(LPCSTR lpszFileName,
        OFSTRUCT FAR *lpOpenBuff,
        int fuMode)
```

donde, *lpszFileName* es una cadena de caracteres que contiene el path y el nombre del fichero que se quiere abrir. *lpOpenBuff* es un puntero lejano a una estructura *OFSTRUCT* que contiene información sobre el fichero. Posteriormente se hablará sobre esta estructura. *fuMode* especifica los atributos con los que se quiere abrir el fichero. Estos valores pueden ser:

Tanto para lectura como para escritura existen dos grupos de funciones según el tamaño de los datos a leer

- ***OF_CANCEL:*** Añade un botón CANCEL con la opción *OF_PROMPT* al mostrar la caja de diálogo si el fichero no existe. Si se pulsa este botón, se devuelve el código de fichero no encontrado.

- ***OF_CREATE:*** Crea un nuevo fichero. Si el fichero existe lo borra. Con esta opción se ignoran todas las banderas para compartir el fichero. Si se quiere compartir, el fichero debe ser cerrado y

Los métodos de E/S de ficheros bajo Windows son quizá la parte que más se parece a la programación para DOS. Cuenta con varias funciones, tanto para abrir como para cerrar, leer o escribir de un fichero. Por otra parte, el API suministra varias funciones para poder acceder a los ficheros de inicialización (.INI), tanto de Windows como los nuestros particulares.

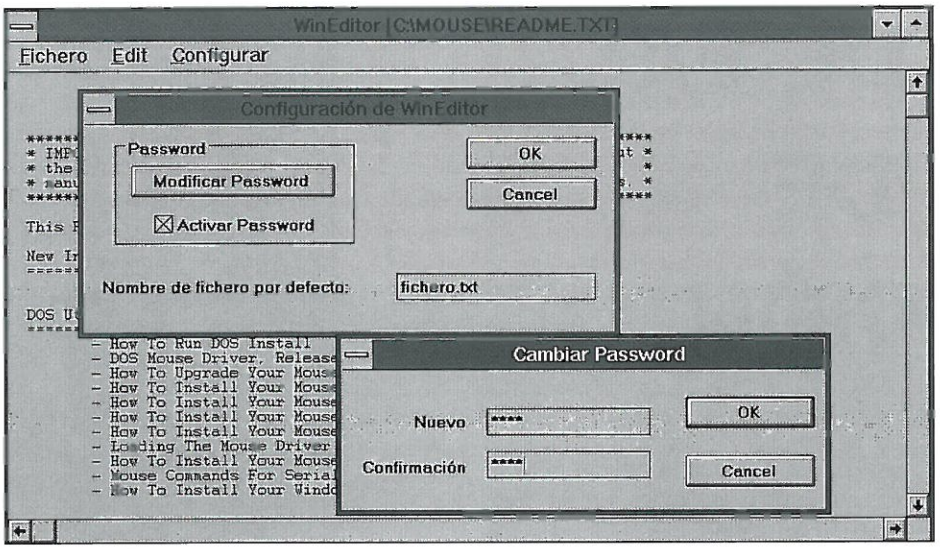


Figura A: WinEditor Versión 1.0.

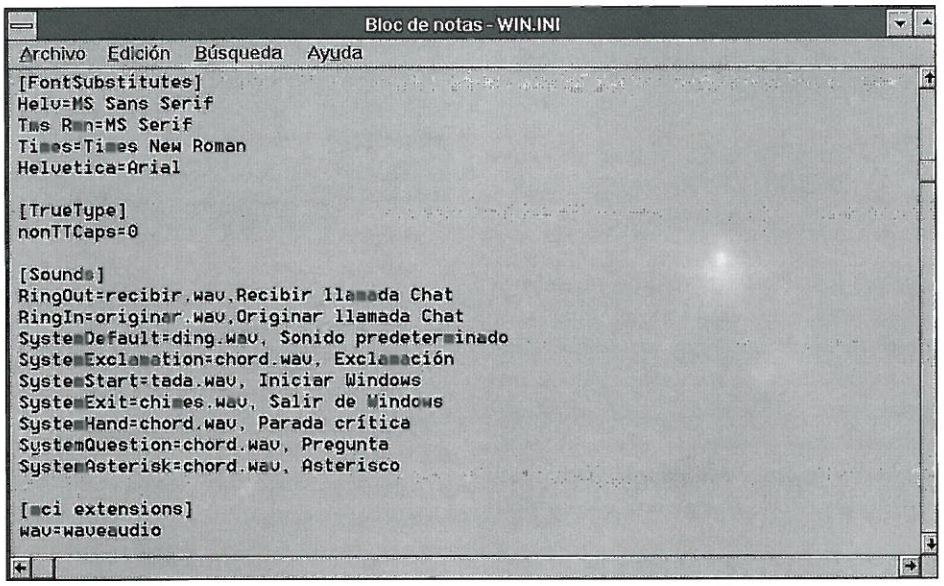


Figura B: Fichero de inicialización WIN.INI.

vuelto a abrir con las opciones para compartir deseadas.

- ***OF_DELETE:*** Borra el fichero indicado.

- ***OF_EXIST:*** Abre y cierra el fichero. Sirve para verificar la existencia del fichero. No actualiza los datos de fecha.

- ***OF_PARSE:*** Llena la estructura *OFSTRUCT* suministrada en el *buffer*, pero no realiza ninguna otra acción.

- ***OF_PROMPT:*** Muestra una caja de diálogo en el caso de que no exista el fichero indicado. Este diálogo informa sobre la no existencia del fichero e insta a insertar un disco en la unidad A con dicho fichero.

- ***OF_READ:*** Abre el fichero sólo con permiso de lectura.

- ***OF_READWRITE:*** Abre el fichero con permiso para lectura y escritura.

- ***OF_REOPEN:*** Abre el fichero utilizando la información del *buffer* para reabrir ficheros.

- ***OF_SEARCH:*** Windows busca el fichero por otros caminos si no se encuentra en el indicado. El método de búsqueda es el siguiente:

1. Se busca en el directorio actual.
2. Se busca en el directorio que contiene WIN.COM (normalmente, C:\WINDOWS).

```
[Sección_1]
Entrada_1 = Valor_1
.
.
Entrada_N = Valor_N

[Sección_2]
Entrada_1 = Valor_1
.
.
Entrada_N = Valor_N
.
.
[Sección_N]
Entrada_1 = Valor_1
.
.
Entrada_N = Valor_N
```

Figura C: Estructura de los ficheros de inicialización.

3. Se busca en el directorio del sistema, normalmente, C:\WINDOWS\SYSTEM.
4. Se busca en el directorio que contiene el ejecutable en uso.
5. Se busca en los directorios listados por la variable de entorno *PATH* de *DOS*.
6. Se busca en los directorios de la red.

- ***OF_SHARE_COMPAT:*** Permite que otros programas puedan abrir el fichero al mismo tiempo un número determinado de veces.

- ***OF_SHARE_DENY_NONE:*** Abre un fichero a compartir sin restricciones de lectura ni de escritura.

- ***OF_SHARE_DENY_READ:*** Abre un fichero a compartir no permitiendo la lectura al resto de los programas.

- ***OF_SHARE_DENY_WRITE:*** Abre un fichero a compartir no permitiendo la escritura al resto de los programas.

- ***OF_SHARE_EXCLUSIVE:*** Abre un fichero a compartir no permitiendo ningún tipo de acceso al fichero por parte de otros programas.

- ***OF_VERIFY:*** Compara la hora y fecha de la estructura *OFSTRUCT* del *buffer* con la hora y fecha del fichero.

Si no coinciden, la función retorna un valor *HFILE_ERROR*.

- ***OF_WRITE:*** Abre el fichero sólo con acceso de escritura.

Como se puede observar, existen muchas formas de abrir un fichero, permitiéndonos un control más intenso del que podíamos hacer bajo DOS. La función devuelve el *handle* del fichero si todo ha sido correcto y *HFILE_ERROR* si no lo ha sido.

Se había quedado pendiente la sintaxis de la estructura *OFSTRUCT*, la cual contiene información sobre el fichero abierto:

```
typedef tagOFSTRUCT
{
 BYTE cBytes;
 BYTE fFixedDisk;
 BYTE nErrCode;
 BYTE reserved[4];
 BYTE szPathName[128];
} OFSTRUCT;
```

donde, *cBytes* es el tamaño en bytes de la estructura. *fFixedDisk* especifica si el fichero está en un disco duro o no. Es cero si no está en un disco duro. *nErrCode* es un código de error de MS-DOS. *Reserved*, uso reservado para uso futuro. *szPathName* contiene el path y el nombre del fichero.

La otra función encargada de la apertura de ficheros es *_lopen*, cuya sintaxis es la mostrada:

```
HFILE _lopen(LPCSTR lpszFileName,
int fnOpenMode);
```

donde, *lpszFileName* es el nombre del fichero (incluido el path). *fnOpenMode* es el modo de apertura. Este valor puede ser combinación de los valores de acceso y los valores opcionales para compartir. Los valores de acceso son:

- *READ*: Acceso de sólo lectura.
- *READ_WRITE*: Acceso de lectura y escritura.
- *WRITE*: Acceso de sólo escritura.

Los valores para compartir son los mismos que los *OF_SHARE...* de la función *OpenFile*. La función devuelve el *handle* del fichero si todo ha sido correcto y *HFILE_ERROR* si no lo ha sido.

Existe una tercera función para crear ficheros que viaja junto con *_open*. Esta función es *_lcreat*, que sirve para crear un fichero. Su sintaxis es:

```
HFILE _lcreat(LPCSTR lpszFineName,
int fnAttribute);
```

donde, *lpszFileName* es como siempre el nombre del fichero. *fnAttribute* es un valor con los atributos para la creación del fichero. Estos valores pueden ser:

0 Normal, se puede leer y escribir.
1 Sólo lectura, no se puede abrir para escribir.
2 Oculto, no se verá al mostrar el directorio.
3 Del sistema, no se verá al mostrar el directorio.

La función devuelve el *handle* del fichero si todo ha sido correcto y *HFILE_ERROR* si no lo ha sido.

CIERRE DE FICHEROS

Existe solamente una función para cerrar un fichero, tanto si ha sido abierto con *OpenFile*, como si ha sido abierto con *_lopen*. Esta función es *_lclose*, cuya sintaxis es la siguiente:

```
HFILE
_lclose(HFILE hf);
```

donde, *hf* es el *handle* del fichero abierto devuelto tanto por *OpenFile* como por *_lopen*. La función devuelve cero si se ha cerrado con éxito o *HFILE_ERROR* si no ha sido así.

ACCESO AL CONTENIDO DEL FICHERO

Tanto para lectura como para escritura existen dos grupos de funciones, según el tamaño de los datos a leer. Estas funciones son:

- *_lread* y *_lwrite* para leer y escribir sobre un fichero datos menores de 64 Kbytes.

- *_hread* y *_hwrite* para leer y escribir sobre un fichero datos mayores de 64 Kbytes. Para ello emplea como *buffer* de lectura/escritura un puntero *huge*.

La sintaxis de *_lread* y *_hread* es la siguiente:

```
UINT _lread(HFILE hf, void _huge*
hpvBuffer, UINT cbBuffer)
long _hread(HFILE hf, void _huge*
hpvBuffer, long cbBuffer)
```

donde, *hf* es el *handle* del fichero del cual se va a leer. *hpvBuffer* es el *buffer* donde se guarda la lectura. *cbBuffer* es el número de bytes a leer.

Como se puede apreciar, la única diferencia en la definición de estas funciones es la limitación que tiene *_lread*, por no poder leer más de 65.535 caracteres, impuesta no por el tipo de puntero del *buffer* sino por *cbBuffer* al ser un entero sin signo (16 bits). Ambas funciones devuelven el número de caracteres leídos o *HFILE_ERROR* si se ha producido algún error.

La sintaxis para *_lwrite* y *_hwrite* es la siguiente:

¿SABE QUE YA PUEDE COMPRAR SU MICROSOFT® WINDOWS 95

¿Sabe que puede convertir su ordenador en una potente máquina gracias a la tecnología de Microsoft Windows 95?

Pídalo ya en los Centros de Actualización Microsoft o en su distribuidor habitual.

Para más información llámenos al telf.: (91) 804 00 96

Microsoft
¿HASTA DONDE QUIERES LLEGAR HOY?

PROG.

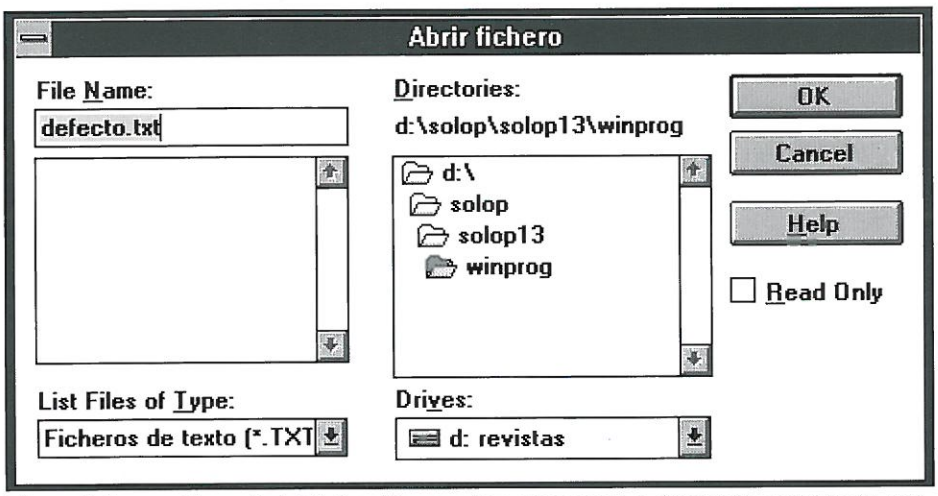


Figura 1. Los archivos de inicialización permiten almacenar el directorio por defecto donde buscar los ficheros.

UINT _lwrite(HFILE hf, void const _huge* hpvBuffer, UINT cbBuffer)
long _hwrite(HFILE hf, const void _huge* hpvBuffer, long cbBuffer)
donde los parámetros tienen el mismo significado que en las dos funciones anteriores, pero para escritura sobre el fichero. El valor devuelto indica los caracteres escritos en el fichero o *HFILE_ERROR* si se ha producido algún error.

POSICIONAMIENTO EN EL FICHERO

Existe también una función de posicionamiento dentro del fichero llamada *_llseek*, cuya sintaxis es:

LONG _llseek(HFILE hf, LONG lOffset, int nOrigin)

donde, *hf* es el *handle* del fichero. *lOffset* es el desplazamiento en bytes del desplazamiento y *nOrigin* indica el origen del desplazamiento. Los valores posibles son:

La estructura OFSTRUCT contiene información sobre el fichero abierto

0 Se mueve desde el origen del fichero.
1 Se mueve desde la posición actual dentro del fichero.

- 2 Se mueve desde el final del fichero.

La función devuelve la posición resultante en bytes desde el origen del fichero o *HFILE_ERROR* si se ha producido algún error.

Figura 2. En un archivo de inicialización pueden almacenarse la ruta por defecto para guardar los ficheros, así como su nombre y tipo.

FICHEROS DE INICIALIZACIÓN

Los ficheros de inicialización surgen para restablecer las propiedades del programa después de que éste haya sido cerrado. Para ello el API de Windows tiene una serie de funciones encargadas de gestionar estos procesos. Un ejemplo notorio de un fichero de inicialización es WIN.INI, donde se guarda toda la información sobre el entorno de Windows. La estructura básica de este tipo de ficheros se ilustra en la Figura C.

Existen dos grupos idénticos de funciones, dependiendo de si queremos escribir en el fichero WIN.INI o bien en un fichero particular nuestro. La pregunta que se plantea es, ¿Cuándo escribir las inicializaciones en WIN.INI y cuándo crear nuestro propio fichero? La respuesta es sencilla, y parte de la base de que sólo se debe de escribir en WIN.INI si las modificaciones afectan a todo el conjunto de ventanas. De no ser así se recomienda crear un fichero particular. Para este ejemplo se ha optado por crear uno particular, primero porque las modificaciones no afectan y segundo por seguridad, ante las posibles pruebas que pueden conllevar un daño irreparable. Si el lector se atreve a modificar WIN.INI, se le recomienda hacer primero una copia de seguridad de dicho fichero.

Las funciones encargadas de escribir y leer en WIN.INI son:

- ***WriteProfileString:*** Escribe una entrada en el fichero.
- ***GetProfileString:*** Lee el valor de una entrada como una cadena de caracteres.
- ***GetProfileInt:*** Lee el valor de una entrada como un número entero.

La sintaxis de las funciones para leer y escribir en un fichero privado en inicialización son las siguientes. Para escribir:
BOOL
WritePrivateProfileString(LPCSTR lpszSection,
LPCSTR lpszEntry,
LPCSTR lpszString,
LPCSTR lpszFilename)

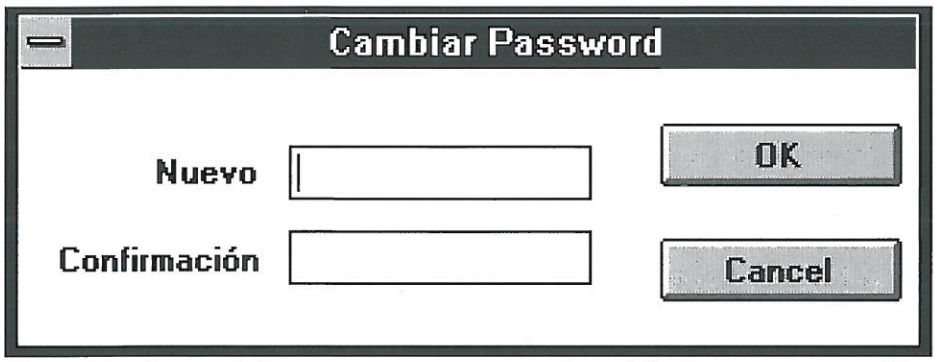


Figura 3. La *Password* de acceso a un programa puede también almacenarse en un fichero de inicialización.

donde, *lpszSection* indica la sección donde queremos escribir, *lpszEntry* la entrada donde queremos escribir, *lpszString* el valor que va a ser asignado a la entrada y *lpszFilename* el nombre del fichero privado donde se escribirá. La función devuelve cero si se ha producido algún error.

Para leer en forma de cadena de caracteres:

```
int GetPrivateProfileString
(LPCSTR lpszSection,
LPCSTR lpszEntry,
LPCSTR lpszDefault,
LPSTR lpszReturnBuffer,
int cbReturnBuffer,
LPCSTR lpszFilename)
```

donde, *lpszSection* es la sección donde se quiere leer, *lpszEntry* la entrada de la que se quiere leer, *lpszDefault* el valor por defecto en caso de lectura errónea, *lpszReturnBuffer* el *buffer* destino de la lectura, *cbReturnBuffer* el tamaño en bytes de la lectura y *lpszFilename* el nombre del fichero privado del cual se leerá.

La función devuelve el número de caracteres leídos sobre el *buffer* sin incluir el nulo.

Para leer en forma de número entero:

```
int GetPrivateProfileInt
(LPCSTR lpszSection,
LPCSTR lpszEntry,
int default,
LPCSTR lpszFilename)
```

donde, *lpszSection* es la sección donde se quiere leer, *lpszEntry* la entrada de la que se quiere leer, *default* el valor entero por defecto en caso de error en la lectura y *lpszFilename* el nombre del fichero privado del cual se leerá. La función devuelve el valor leído de la entrada.

Los ficheros de inicialización surgen para restablecer las propiedades del programa después de que éste haya sido cerrado

OpenFile nos permite disponer de información sobre el fichero abierto o mostrar automáticamente un diálogo de error si un fichero no ha sido encontrado

WINEDITOR 1.0

En esta última entrega de nuestro editor se da funcionalidad a las capacidades de configuración. Esto se refleja en el principio del código (función WinMain) cuando cargan los datos de inicialización con las funciones antes expuestas. Como se puede apreciar, el código es realmente sencillo, pero cabe destacar la llamada a la función *WritePrivateProfileString* con todos los parámetros a *NULL* excepto el nombre del fichero. Esta llamada se realiza después de escribir en el fichero, y es debido a que Windows mantiene una copia en memoria del fichero para mejorar prestaciones, y la forma de actualizar el disco es llamando de esta guisa a dicha función. También resaltar al lector que vea en el procedimiento de gestión de mensajes del diálogo de configuración el modo de actualizar el fichero de inicialización.

Por otra parte, todo lo referente a la parte de E/S de ficheros ya estaba incluida en la versión anterior del programa, si bien se ha detallado toda esta funcionalidad en este artículo.

El fichero de inicialización se crea por primera vez por el propio programa, y residirá en el directorio donde esté WIN.INI.

CONCLUSIÓN

En este artículo se ha pretendido dar una idea de los métodos que existen para abrir, modificar y cerrar un fichero, utilizando las funciones del API de Windows, así como ver todas las opciones que nos brindan. Por otro lado, también se pretende que el lector sea capaz de crear y modificar su propio fichero de inicialización.

CREACIÓN DE UN MENÚ DE DESARROLLADOR

José María Peco

Hace tiempo que vengo examinando los catálogos de shareware, y he tenido ocasión de convencerme de que son muchos los usuarios de PC que prefieren hacer uso de un menú para gestionar sus programas de aplicación y utilidades en lugar de escribir cada vez el comando a ejecutar; y, por otra parte, al llevarlo a la práctica, también he podido comprobar que son innumerables las ventajas que esto acarrea, pues no se deja a la memoria humana la gestión de los nombres de los programas ni los parámetros necesarios para ejecutar un programa determinado.

En los grandes sistemas ocurre algo parecido, solo que por regla general es el personal del departamento de sistemas quien se encarga de preparar los paneles o menús encadenados que permitirán al desarrollador llamar a las distintas utilidades del sistema, por lo que a éste último sólo le queda el recurso de usar lo que otros han hecho para él, o usar la línea de comando para escribir cada vez el comando que desea ejecutar con todos sus parámetros.

El presente articulo pretende conducir al programador a personalizar su gestión de desarrollo en un gran sistema, entendiendo por tal a las maquinas que soportan los sistemas operativos: MVS en IBM y VMS en DIGITAL.

IBM: PANEL DE SELECCIÓN DE ISPF BAJO TSO

En este entorno, el sistema operativo es el MVS, y el desarrollador para poder hacer uso de esta utilidad debe estar trabajando bajo TSO, pues de esta forma podrá invocar los servicios de ISPF que permiten ejecutar un panel de selección o menú.

Para poder hacer uso de esta utilidad en MVS debe trabajarse bajo TSO

El ejemplo que va a servir de hilo conductor de esta parte del artículo, utiliza como lenguaje de comandos el lenguaje de CLIST que acompaña al sistema operativo, si bien puede usarse cualquier otro lenguaje como mostraremos en próximos artículos.

DEFINICIÓN DEL PANEL

La definición del panel del ejemplo se muestra en la figura 1. A la vista de su definición, se puede distinguir en ella las 5 secciones que se estudian a continuación, todas ellas precedidas del símbolo ')':

)ATTR

Esta sección define los caracteres que tienen un significado especial en este panel. Así, los caracteres definidos para las clases genéricas por defecto son :

(%) Carácter que está asociado al atributo de texto intensificado.

(+) Carácter que define el comienzo de texto normal.

(~) Carácter que define que la variable que le sigue es de entrada.

)BODY

Esta sección define la imagen de la pantalla, casi como la vera el usuario cuando se ejecute el panel, y digo 'casi' ya que en esta sección inter-

```
)ATTR DEFAULT(%+~)
)BODY
+------------------------------------------------------%MENU          PRINCIPAL          DE
JMP+------------------------------------------------------------------------------------
%OPCION ===>~ZCMD                                                                          +
%
%
%                                                                                  +
%                    1 +Listar librerias alocadas                                  +
%                    2 +Tratar librerias alocadas                                  +
%                    3 +Editar fichero                                             +
%                     4 +Programar PFs                                             +
%                    5 +Menu utilidades auxiliares                                 +
%                       +
%                    X +SALIR DEL MENU                                             +
%                                                                                  +
%                                                                                  +
)INIT
 &ZPRIM = YES
  .HELP = USRH100
)PROC
&ZSEL = TRANS(TRUNC(&ZCMD,'.')
          1,'CMD(LISTA)'
          2,'CMD(EX "DESJMP.UTILIDAD(JMPALO00)")'
          3,'PANEL(ISREDM01)'
          4,'PGM(ISPOPT) PARM(ISPOPT3)'
          5,'PANEL(MENU0010)
          X,'EXIT'
          ',',' '
         *,'?' )
)END
```

Figura 1: Definición del plano.

```
------------------------------------------------MENU        PRINCIPAL        DE        JMP
------------------------------------------------------------------------------------
 OPCION ===>

          1    Listar librerias alocadas
          2    Tratar librerias alocadas
          3    Editar fichero
          4    Programar PFs
          5    Menu utilidades auxiliares

          X  SALIR DEL MENU
```

Figura 2: Aspecto del menú principal de JMP.

vienen los caracteres definidos en la sección anterior, y los nombres de las variables, tal y como puede apreciarse en la definición adjunta.

La primera línea, se compone sólo de texto, guiones con atributo normal, literales en intensificado, hasta que aparece el carácter '+' que define para el resto de la línea (guionado final) el atributo de texto normal.

La segunda línea contiene un texto en intensificado, delimitado por el carácter que define el comienzo de la variable de entrada ZCMD. La longitud de esta variable viene limitado por la posición del carácter delimitador "+"

El resto de líneas de esta sección sólo presentan el numero de la opción en intensificado y lo que representa cada opción con atributo normal.

)INIT

Esta sección define los procesos que se ejecutarán antes de

Update now!

WATCOM

Compiladores para lenguajes profesionales
- Lenguaje C/C++V.10.0 CD ROM
- Lenguaje Fortran 77/32 Bits v 9.5

Ambiente de desarrollo visual (Front End)
- Lenguaje OS/2 Rexx
- VX-REXX v.2.1 Standard Edition
- VX-REXX v.2.1 Client/Server

Bases de datos relacionales (llamada SQL) v 4.0
- Plataformas: Windows, NT, OS/2, DOS y Netware
- Usuarios: Standalone - Network Server
- Lenguaje: C/C++

RAIMA

Raima Database Manager: Base de Datos
- Lenguaje: C
- Sistema Operativo: DOS, Windows, OS/2, UNIX System V, Berkeley 4.2 AIX, Sun OS, SCO, QNX, YULTRIX y VMS
- Plataformas: DOS, OS/2
- Modalidades: Module y System
- Royalty Free

VELOCIS

Velocis: Base de datos. Tecnología Cliente/Servidor
- Lenguaje C/C++
- Sistema Operativo: Windows, NT y UNIX
- Plataformas: Windows, NT, OS/2, Netware 386 NLM, OS/2 y UNIX
- Velocidad de acceso 1/3 más rápido que otras B/Datos

PHARLAP

Gestiona la memoria optimizando los límites de todas las versiones DOS

SEQUITER Software Inc.

Librerías de programas para trasladar aplicaciones de Dbase a C/C++
- CodeServer v.5.1 Cliente/Servidor

OFERTA ESPECIAL

Para los lectores de SÓLO PROGRAMADORES

Infórmate llamando:
Tel.: (93) 225 39 95
Fax: (93) 225 40 03

CMR Competitive Manufacturing Research

C. Lope de Vega, 21 bjs.
08005 Barcelona

presentar el panel, y que utilizaremos para cargar las siguientes variables:

ZPRIM = YES

carga en la variable ZPRIM del sistema el valor YES, con lo cual se le indica al sistema que éste es un panel raíz. Esto significa que cuando, estando en niveles inferiores, se especifique *=5.3* , el sistema entenderá que partiendo de este panel raíz se desea ejecutar la opción 5, la cual nos lleva a otro panel de selección o menú, del cual se elige la opción 3. Como es de suponer, esta opción solo se especifica en el caso de que se desee que sea 'raíz'.

.HELP = JMP001P

carga la variable del sistema .HELP con el nombre del panel que servirá de ayuda cuando se ejecute el comando HELP (normalmente asociado a la PF1)

)PROC

Esta sección define los procesos que se ejecutarán después de pulsar la tecla INTRO. Al ser un panel de selección normalmente hará uso de las siguientes funciones:

variable1=TRUNC(variable2,'caracter')

Esta función carga en *variable1* el valor obtenido al truncar la *variable2* por el carácter especificado como segundo argumento de la función.

variable3 = TRANS(variable4 par1 par2 ...)

donde *par1* representa la pareja *codigo1,valor1* *par2* representa la pareja *codigo2,valor2* etc.

Esta función devuelve en *variable3* el valor_x asociado al código_x del par_x que coincida con el valor especificado en *variable4*.

Combinando ambas funciones, y suponiendo que se haya escrito el valor 5.3 en la variable de entrada de la selección ZCMD, resulta lo siguiente:

La función TRUNC devuelve el valor 5 pues trunca por el punto.

La función TRANS busca entre los distintos pares especificados aquel que coincide con el valor 5, por lo que devuelve 'PANEL(MENU0010)' como valor de la variable *ZSEL*.

Cada panel debe ser un miembro de una librería de paneles asignada previamente

En el ejemplo propuesto, figuran dos pares especiales, cuyo significado es el siguiente:

' ',' ' Este par especifica que si no selecciona nada, no devuelva nada, por lo que indirectamente se especifica que se mantenga en el mismo panel.

*,'?' Este par especifica que si el valor especificado en variable4 no coincide con ningún código previo, muestre el mensaje de OPCIÓN INVALIDA.

En el ejemplo presentado, la opción 1 invoca al comando de TSO: LISTA que presenta todas las librerías que se encuentran asignadas (alocadas) a nuestro entorno.

La opción 2 invoca a la clist que se estudio en el número de julio, y que permite tratar de forma automática las librerías del entorno haciendo uso del comando anterior.

La opción 3 invoca al panel del editor del ISPF

La opción 4 invoca un programa del sistema, concretamente al programa que permite programar las PF's.

La opción X retorna al menú anterior.

)END

Esta sección es una sección vacía, es decir que sólo especifica el final de la definición del panel, y no puede contener ningún argumento.

CÓMO INVOCAR AL PANEL

Cada panel debe ser un miembro de una librería de paneles asignada previamente. Para lo cual, si suponemos que la definición del panel que se muestra en la figura 1 tiene por nombre DSN=DESJMP.PANELES(MENURAIZ) puede ejecutarse la clist siguiente, que debe ser escrita en mayúsculas:

```
/* **** clist : MENURAIZ ******** */
PROC 0

ISPEXEC LIBDEF ISPPLIB +

    DATASET ID('DESJMP.PANELES')
ISPEXEC SELECT PANEL(MENURAIZ)
EXIT
/* ****************************** */
```

Y la anterior clist, suponiendo que tenga por nombre DSN=DESJMP.JMP.CLIST puede invocarse directamente desde línea de comando, escribiendo:

TSO EXEC 'DESJMP.JMP.CLIST'

para que aparezca el panel de la *figura 2;* aunque si el usuario tiene por prefijo DESJMP bastaría con escribir en la línea de comandos para ejecutar la misma clist:

TSO EX JMP

Y, como el espacio no perdona, dejo para el próximo mes la confección de un menú en DIGITAL, no obstante, para aquellos que quieran intentarlo, en el disquete que acompaña a la revista , se adelanta el fuente de dicho menú.

CORREO LECTORES

Para cualquier duda referente a los temas/artículos tratatos en la revista, escriba una carta a:

Sólo Programadores
Referencia: *Correo Lectores*
TOWER COMMUNICATIONS
C/ Marqués de Portugalete, 10 Bajo
Madrid 28027

COMO SUSCRIBIRSE A

Suscríbase enviando este cupón por correo o fax (91) 320.60.72, o llamando al teléfono (91) 741.26.62 Horario 9 a 14 y 15:30 a 18:30 h.

Deseo suscribirme a la revista SÓLO PROGRAMADORES acogiéndome a la siguiente modalidad:

☐ Suscripción: 1 año (12 números) **por sólo** 11.950 ptas. (ahorro 20%). ☐ Estudiantes carreras técnicas: 8.950 ptas. (ahorro 40%)

*** ESTA OFERTA ANULA LAS ANTERIORES, DESCUENTOS NO ACUMULABLES.**

Nombre y apellidos.....Domicilio.....

Población.....C.P .....Provincia.....Telf.....Profesión.....

FORMA DE PAGO:

☐ Con cargo a mi tarjeta VISA nº ____ ____ ____ ____

Fecha de caducidad de la tarjeta.....Nombre del titular, si es distinto.....

☐ Domiciliación bancaria.
Señor Director del banco .....
Población .....
Ruego a vd. que se sirva cargar en mi ☐ cuenta corriente ☐ libreta de ahorro número.....
el recibo que le será presentado por TOWER COMMUNICATIONS, S.R.L.
como pago de mi suscripción a la revista SÓLO PROGRAMADORES.

CODIGO CUENTA CLIENTE

ENTIDAD	OFICINA	DC	Nº CUENTA

Firma:

☐ Contra-reembolso del importe más gastos de envío.
☐ Cheque a nombre de TOWER COMMUNICATIONS S.R.L., que adjunto.
☐ Giro Postal (adjunto fotocopia del resguardo).

Rellena este cupón y envíalo a:
TOWER COMMUNICATIONS SRL
C/ Marques de Portugalete 10, Bajo
28027 Madrid.

TCP/IP EN LÍNEAS SERIE

David Aparicio

TCP/IP ha sido siempre el estándar de conectividad entre ordenadores Unix. Hoy día, se está extendiendo al resto de sistemas operativos. La red Internet, donde las conexiones se basan en el protocolo IP, es la mayor demostración de esta estandarización "de facto". Veamos algunos aspectos básicos del mismo.

El medio por excelencia para explotar las conexiones IP es una red ethernet. Normalmente, se trata de un cable coaxial al que se conectan varios nodos, mediante tarjetas que comunican sus datos a 10 Mbits por segundo, como podemos observar en la figura 1. Este sistema, tan corriente en los entornos empresariales, es impracticable en el hogar, donde hay un sólo nodo. Sin embargo, la popularidad de los módems, que permiten conectar mediante el teléfono a otros sistemas, abre un nuevo mundo de posibilidades.

La forma en la que se suele emplear una conexión por módem es la denominada "dedicada" o "terminal". Una vez que se conecta con el sistema remoto, se presenta una pantalla que nos permite realizar una operación cada vez. Lo ideal, sin embargo, es "multiplexar" la conexión, es decir, emplear algún sistema que nos permita mantener varios diálogos simultáneos por el mismo cable. Esto nos lleva al protocolo que nos va a ocupar hoy: el estándar IP y su implementación en líneas serie, tanto SLIP como PPP.

REUNIENDO LOS COMPONENTES

Las comunicaciones por red, como todo tipo de diálogo con periféricos Unix, requiere dispositivos en los que poder leer y escribir. Si el núcleo de Linux de su ordenador se compiló con soporte de TCP/IP, podrá observar los dispositivos disponibles. Teclee:

```
cat /proc/net/dev
```

Aparecerá una lista con todos los dispositivos de red con soporte en el sistema. Para seguir este artículo, necesitará tener los siguientes:

- En primer lugar, nos encontramos con el dispositivo '*lo*' (*loopback*). Podemos interpretarlo como un enlace de pruebas. Cualquier sistema con IP utiliza este dispositivo para referirse a sí misma y también para realizar pruebas internas.
- Por otro lado podemos encontrarnos con los dispositivos '*sl0*' hasta '*sl3*'. Esto indica hasta cuatro dispositivos de tipo SLIP, y representan los enlaces de este tipo que podemos realizar con cada una de las líneas serie de nuestro PC.
- Los dispositivos '*ppp0*' hasta '*ppp3*' representan enlaces de tipo PPP que podríamos realizar con las mismas líneas serie. Al igual que antes, es necesario disponer de un núcleo que haya sido compilado con este soporte. Por el momento, no se preocupe por tanta sigla. Ya se verán más adelante.
- Pueden aparecer otros dispositivos, como '*eth0*', pero sólo si su ordenador dispone de alguna placa ethernet y Linux la ha detectado. Todo lo que se verá a continuación se aplica también a este tipo de dispositivos, pero no nos ocuparemos de ellos en el presente artículo.

Si no dispone de dispositivos *sl* o *ppp*, necesita rearrancar con un núcleo que tenga dicho soporte incluido.

Para seguir, necesitaremos instalar ciertos ejecutables (*dip* y *pppd*) en nuestro sistema:

DIP es un programa que permite automatizar la conexión con otro sistema, e incluye el protocolo SLIP.

PPPD controla la parte de interacción con el usuario en la implementación del protocolo PPP (la otra parte está compilada en el núcleo). En un futuro, DIP podrá manejar de forma autónoma tanto

SLIP como PPP, pero por el momento necesitamos ambos paquetes.

Las versiones que se distribuyen en Slackware (2.2) han producido ciertos problemas de configuración durante el desarrollo de este artículo, de modo que se incluyen actualizaciones de ambos programas (*ppp 2.1.2c* y *dip 3.3.7n*), con las cuales se han realizado satisfactoriamente todas las pruebas. Para instalarlos, asegúrese de haber borrado versiones previas (con *pkgtool*), y descomprima el archivo proporcionado en el disco de la revista:

```
cd /
tar xvfz /path/slip.tgz.
```

Los programas como *ping*, *telnet* y *ftp*, que vamos a emplear una vez que tengamos funcionando la comunicación IP, se encuentran en la serie N de Linux. Si no instaló esta serie, quizá ha llegado el momento de hacerlo antes de proseguir. Recuerde que no debe incluir los paquetes '*dip*' ni '*ppp*' que se proporcionan con Slackware, puesto que están algo desfasados. Sin embargo, puesto que es interesante la documentación que acompaña a ambos paquetes, se recomienda descomprimirlos en un subdirectorio aparte (no en el raíz) para poder realizar consultas, pero evitando que se puedan ejecutar los programas obsoletos.

Comentemos aquí que justo antes de acabar la instalación se le preguntará si va a emplear sólo el dispositivo '*loopback*'. De alguna forma, lo que se está queriendo saber es si usted dispone de conexión TCP/IP mediante placa ethernet, para que el sistema se autoconfigure cada vez que arranque. En la mayor parte de casos, y como no dispondremos de dicha conexión, la respuesta será si usar sólo el dispositivo '*loopback*'.

CONCEPTOS BÁSICOS

Las comunicaciones TCP/IP requieren que cada nodo se identifique con un número de 32 bits. A partir de ahora denominaremos *dirección IP* a dicho número. Se representa mediante cuatro números de 8 bits en decimal, separados por puntos. Por ejemplo, el número 1.0.4.147 es una dirección IP.

Normalmente, se asigna a cada nodo una dirección IP por cada dispositivo de red que tenga en funcionamiento.

Todas las máquinas que tienen un sólo dispositivo para comunicarse (aparte del de *loopback*), se denominan nodos simples o "puntos". Estos nodos tienen una sola dirección IP que los identifica.

Las máquinas que tienen varios dispositivos de red se denominan *pasarelas*, o *gateways*. Estos sistemas suelen disponer de varias direcciones IP, cada una de las cuales los identifica en una red.

Todas las direcciones IP descomponen sus 32 bits en una parte denominada "de red" y otra "de nodo". La parte de dirección que corresponde a cada una depende del primer número (8 bits) de la dirección IP, según la figura 2.

A un cable determinado, sólo hay conectados ordenadores en cuya dirección IP coincide el número de red. El límite de nodos conectables a dicho enlace viene dado por el tamaño de la parte de dirección que indica el número de nodo. Por ello, para comunicar dos máquinas cuya dirección de red no coincide, como se encuentran conectados a cables diferentes, necesitaremos trasvasar los mensajes de una red a otra. Esta función la realizan los nodos pasarela, en base a una tabla interna en la que deciden el dispositivo por el que encauzar los paquetes que tengan cierta dirección IP como destino.

A lo largo de los ejemplos que se mostrarán a partir de aquí, siempre haremos referencia a dos nodos:

- *gárgola*: Conectado por ethernet a una red de tipo A, con dirección IP 1.0.4.7. Como servidor de PPP y de SLIP, es pasarela de algunos nodos de esta red.
- *fénix*: Cliente en conexiones PPP y SLIP, es un nodo simple con dirección IP fija (1.0.4.248) en una red de clase A a la que conecta por módem.

Observe el esquema de la figura 3, para hacerse una idea de la configuración que servirá de base para nuestros ejemplos.

INTERFACES Y RUTAS

El primer programa a tratar es *ifconfig*. Mediante él, se asocia un dispositivo de red (los que se encuentran en */proc/net/dev*) con una dirección IP, se definen algunas características del enlace, y se permite activar o desactivar dicho dispositivo.

Los dispositivos del tipo de una ethernet se denominan de multidifusión (*broadcast*). Por la propia naturaleza de éstos, cualquier información depositada aquí es escuchada por todos los nodos que se encuentren conectados al mismo cable.

Por otro lado, los dispositivos de línea serie se denominan de punto a punto (*point-to-point*). En éstos, sólo hay dos interlocutores en cada cable. Lo que uno envía, el otro lo recibe.

Teclee:

```
ifconfig
```

Debería mostrarse una lista de todos los dispositivos que tienen dirección asociada, y que se encuentran activos (*up*). Lo normal es que sólo vea el dispositivo '*lo*', con dirección IP asociada 127.0.0.1.

Es posible examinar las características de un dispositivo, aunque esté inactivo (*down*). Teclee:

```
ifconfig ppp0
```

Si dispone del soporte del núcleo para protocolo PPP, en este momento puede observar las características de este enlace. Si no es así, debe disponer de un núcleo con soporte PPP y SLIP, para poder seguir los ejemplos. Obténgalo de la serie Q o recompile el núcleo.

En el interfaz ppp0, se indican algunas de sus características, como "punto a punto", pero (por el momento) no hay dirección física asociada.

Si se ha fijado, aparece otro concepto en la respuesta de este comando. La máscara de red (*netmask*) indica un número donde la parte de la dirección IP que queremos que corresponda con el "número de red" está a unos, y la parte del "número de host" está a ceros. Este número permite al núcleo de TCP/IP distinguir, de forma eficiente, ambos componentes de la dirección IP. Por ejemplo, en el dispositivo *lo*, con dirección IP 127.0.0.1 (lo que corresponde a una red tipo A) la máscara es "255.0.0.0".

En la figura 2 se observa que, del número de *hosts* disponibles para cada red, se restan siempre dos posibilidades. En cualquier dirección IP asignada a un interfaz, la codificacion "todo el campo de host a uno" indica la dirección de multidifusión para esa red. Para el interfaz '*lo*', la dirección de multidifusión será 127.255.255.255. Cuando se envía información con esta dirección destino, la reciben todos los nodos conectados a dicha red. La codificación "todo el campo de host a cero" indica la dirección de la propia red, y tiene valor únicamente en las tablas de encaminamiento, como veremos ahora. Teclee:

route

En su pantalla debería aparecer una asociación entre números de red, la máscara que se aplica en dicha red, si hay un encaminador para llegar a esa red, y el dispositivo por el que hay que enviar cada mensaje para que llegue a su destino.

Como mínimo, aparecerá una línea que indica que las direcciones destino 127.0.0.0 (la red tipo A que identifica el núcleo IP de cada nodo) se encaminan por el dispositivo 'lo', aplicando la máscara 255.0.0.0. En otras palabras, y como se ha comentado antes, todos los mensajes que tienen 127 en el primer campo, se envían por el dispositivo interno al núcleo de TCP/IP.

Por último, aparece una dirección de red *default*, en la primera columna. Esto indica que todos los mensajes que no se puedan encaminar mediante alguna entrada explícita, se dirigen a otro nodo más inteligente por el dispositivo asociado con esta entrada, impidiendo así que el mensaje se pierda. En el ejemplo de la figura, todos los paquetes cuyo destino no sea un componente de la red 1.0.0.0 (la ethernet) son encaminados a la pasarela "ulises", la cual consultará su propia tabla de encaminamiento para saber el destino de los paquetes.

Resumiendo:

- *ifconfig* asocia una dirección IP de nodo con un dispositivo. Indicamos la dirección IP que identifica a nuestra máquina en la red a la que está conectada mediante dicho dispositivo, para que pueda recoger los mensajes dirigidos a dicha dirección.
- Con *route* estamos indicando por qué dispositivo se deben enviar cada rango de direcciones IP. Esto permite escoger el camino (o ruta, de ahí el nombre) por el que deben enviarse los mensajes. Si no hay camino por el que enviar algún paquete (porque no haya una entrada adecuada para dicha dirección en la tabla), éste se tira.

Puede examinar el fichero */etc/rc.d/rc.inet1*, donde se encuentra la configuración que se realiza en el arranque del sistema mediante estos comandos. Este fichero se ha creado en la instalación de la serie N, cuando se le ha preguntado sobre si sólo iba a utilizar el dispositivo *loopback*.

EL PROTOCOLO SLIP

Llegado este punto, y suponiendo que usted dispone de módem y tiene compilado un núcleo con la opción de SLIP activada, veremos cómo conectar con otro nodo mediante paquetes IP, consiguiendo así múltiples operaciones simultáneas sobre una misma línea.

El protocolo toma su nombre de *Serial Line IP*, y supone que tendremos un programa en cada punto de la línea que se encargue de enviar la información codificada a su interlocutor, de forma que se pueda aprovechar la línea para varios diálogos simultáneos.

Supongamos un ejemplo que nos ayude a aclarar este concepto. Dos personas que hablan por teléfono disponen de una línea "dedicada". Es decir, tanto si hablan como si están en silencio, sólo ellos están manteniendo una conversación. Hasta que no cuelguen, no se podrá comunicar desde ninguno de ambos teléfonos con otro interlocutor distinto.

Por el contrario, supongamos dos buzones de correos, situados cada uno en un barrio. Varias personas pueden escribir cartas a cada lado, y echarlas al mismo buzón. Incluso una misma persona puede enviar varias cartas a diferentes personas del barrio donde se encuentra el otro buzón. Sea como sea, hay una persona dedicada (el cartero) que es responsable de recoger toda la correspondencia de un buzón y la deposita en los buzones de las casas de los destinatarios, leyendo el destino en los sobres. Con sólo dos buzones, se pueden comunicar varios grupos de personas simultáneamente.

El cartero de nuestro caso es el programa que toma la información de cada lado y la envía al otro, permitiendo que haya "correspondencia simultánea".

En primer lugar, debemos instalar un programa que nos permita automatizar las operaciones de establecimiento de la conexión y diálogo. Si ha descomprimido correctamente el archivo que acompaña a este artículo (*slip.tgz*), ya tiene disponible el ejecutable *dip*. Este programa se encarga de programar las características de la línea serie, realizar el diálogo de marcado y *login* en el ordenador remoto, y lanzar finalmente el programa que realiza la codificación y decodificación de paquetes SLIP.

Además, hay otro programa, denominado *diplogin*, que se encarga de aceptar conexiones remotas, y lanzar SLIP en el nodo llamado. Veamos primero la configuración de este último.

EL SERVIDOR SLIP

La operación de marcado y login se realiza de la misma forma que cuando efectuamos una llamada con el clásico programa de comunicaciones. Sin embargo, en el fichero */etc/passwd* debemos introducir una línea similar a:

slip::40:14:SLP:/tmp:/usr/sbin/diplogin

Observe que el directorio de entrada es */tmp*. Se suelen indicar así las entradas en las que no se invoca una *shell* normal, como es el caso que nos ocupa. En su lugar, cuando aparece la carátula de bienvenida del sistema llamado, e introducimos *slip* como login, estamos invocando a *diplogin*. En este caso, no incluimos *password*, para facilitar la configuración. Una vez completada, es recomendable establecer alguna.

Una vez invocado, el programa busca en el fichero */etc/diphosts* una línea donde coincida el primer campo con el login (*slip*) que hayamos introducido. Por ejemplo:

slip::fenix:::Entrada slip:SLIP,296

Esto dice: "si el identificativo de login ha sido *slip* y el ordenador que llama es 'fenix', arrancar el protocolo SLIP por

esta línea serie". Si *diplogin* no encuentra una combinación de nombre de nodo remoto y login introducido, cierra la conexión inmediatamente. Esto protege el acceso al sistema, puesto que un posible pirata necesitaría averiguar la combinación de ambos nombres, además de la *password* de entrada. Si la operación de autentificación tiene éxito, se arranca el interlocutor SLIP en el lado del servidor. Se terminará la conexión (y se volverá a crear un *getty* que espere una nueva llamada) al perder la señal *carrier detect* en el puerto serie donde se conecta el módem una vez que el cliente cuelgue.

Ésta es toda la instalación necesaria (además de tener un 'getty' lanzado sobre la línea a la que se conecta el módem, tal y como se explicó en el artículo del mes de junio) para configurar SLIP en el nodo llamado.

EL CLIENTE SLIP

En el nodo que efectúa la llamada (fénix), necesitaremos *dip* y un texto que automatice la entrada. Veamos un ejemplo, que nos permitiría conectar a un nodo remoto (gárgola), y entrar con login *slip*, sin *password*.

```
# Ejemplo de script DIP: slip.dip

# Inicializar nombres de nodo local y
# remoto
  get $local fenix
  get $remote gargola

# Conectaremos a una red A (1.0.0.0)
  netmask 255.0.0.0

# Inicializar línea /dev/modem y
# velocidad 19200
  port modem
  speed 19200

# Marcar número
  send 1234567\r
# Si no aparece "login:" en 45 segundos,
# hay un error
  wait login: 45
  if $errlvl!=0 goto error
# en caso afirmativo, introducir login
  send slip\r

# hemos conectado, lanzar modo SLIP
  default
  print Conexión realizada...
  mode SLIP
  exit

error:
# Hubo algun error, abortar
  print Problema en la conexión
  quit
```

Como podemos observar, la mayor parte del esfuerzo del texto se dedica a completar la conexión y realizar el login. Una vez hecho esto, invocar el modo SLIP se realiza en una simple línea. De hecho, la mayor complejidad de realizar un fichero de automatización reside en lo primero. Cuando todo esté correctamente configurado, la conexión es tan sencilla como teclear:

```
dip slip.dip
```

El fichero *slip.dip* debería entonces realizar todas las operaciones necesarias de forma automática. Si la conexión se realiza con éxito, podremos utilizar todos los programas que se emplean en TCP/IP, como:

```
ping gárgola
ftp gárgola
telnet gárgola
```

Si encuentra algún problema al crear el fichero de entrada para *dip*, debe saber que hay un modo de lanzar el programa en "modo depuración". Teclee:

```
dip -v slip.dip
```

Se deberían visualizar todas las operaciones que *DIP* realiza. Adicionalmente, puede controlar lo que se envía con 'print', añadiendo la siguiente línea en *slip.dip*:

```
echo on
```

Una vez que consiga establecer la conexión con éxito (ármese de paciencia, porque es una operación que puede requerir varios intentos), podrá observar la verdadera potencia de los protocolos IP en líneas serie. Pruebe a abrir varias ventanas (caso de estar en X Windows), o intente conexiones simultáneas desde varias pantallas de consola. Comprobará que puede establecer tantas conexiones como desee.

Para dar por terminada la conexión, teclee:

```
dip -k
```

Esto mata al cliente SLIP, con lo que el módem cuelga y el servidor SLIP también finaliza.

EL PROTOCOLO PPP

Si ya disponemos del protocolo SLIP, ¿para qué nos sirve otro protocolo IP para líneas serie?. Bien, como todas las cosas, cada sistema tiene ciertas ventajas e inconvenientes.

- El protocolo SLIP es más antiguo, y por tanto está más extendido. Casi todos los sistemas que proporcionan conexión IP por línea serie soportan con certeza este protocolo. OS/2 Warp es un claro ejemplo de este argumento. Si usted va a conectar con un servidor de IP, con casi total seguridad soporta SLIP, y, con menor frecuencia, PPP.
- Por otra parte, PPP es mucho más robusto a fallos. En los paquetes que se envían por la línea, viaja un código de detección de errores que permite evitar corrupción en los datos, y está prevista la retransmisión automática de los datos dañados.
- Además, PPP permite compresión de datos de forma nativa, y proporciona encriptación opcional de los datos, de forma que sea más difícil para personas enamoradas de lo ajeno interceptar o entrar ilegalmente en un sistema.
- Este protocolo (PPP) no sólo permite conectar dos sistemas mediante IP, sino que está capacitado para aceptar otros protocolos, como AppleTalk.
- Por último, PPP es el sistema llamado a ser el estándar, que tarde o temprano se impondrá en todos los sistemas, desplazando al protocolo SLIP.

En Linux, el programa encargado de controlar la línea serie se denomina *pppd*. Por cierto, las siglas corresponden a *Point to Point Protocol*. Como en el caso anterior, se necesita algún programa que realice la función de marcado, establecimiento de la conexión, login y lanzado del programa en el interlocutor de forma automatizada. De nuevo DIP cumplirá esta función. Como vamos a ver, desde el punto de

vista del establecimiento de la conexión, ésta es prácticamente idéntica.

EL SERVIDOR PPP

Como en el caso de SLIP, introduciremos en */etc/passwd* la siguiente línea:

ppp::41:14:PPP:/tmp:/usr/etc/pppd

Como en el ejemplo anterior, se toma el directorio */tmp* como base, invocando un programa diferente de la *shell* de una entrada tradicional. En este caso, debemos tener un fichero de configuración, en */etc/ppp/options*, que programa ciertas características en el comportamiento de PPP:

```
asyncmap 200A0000
crtscts
mru 296
netmask 255.0.0.0
-pap
-chap
proxyarp
passive
silent
modem
gargola:fenix
```

La opción *asyncmap* permite indicar un mapa de los caracteres que deben codificarse de modo especial para evitar que sean confundidos por otros protocolos que usen la línea. El número del ejemplo suele funcionar correctamente en TCP/IP.

crtscts y *modem* indican que se establece control hardware de la línea serie, siempre que el cable utilizado lo permita.

Las opciones *pap* y *chap*, con un guión delante, deshabilitan los protocolos de encriptación y protección de datos. Por el momento, y hasta que no se configure correctamente el caso más sencillo, no producirán más que problemas. Cuando se sienta con fuerzas, consulte la documentación que se encuentra en la distribución Slackware para saber más sobre estas opciones de protección.

Las opciones *passive* y *silent* indican que este lado esperará a que el interlocutor comience la negociación de las características de la conexión. Como ocurre en la vida real, en una conversación se requiere que uno de ambos lados empiece a hablar, y el otro escuche, hasta que ambos se pongan de acuerdo sobre las opciones del diálogo. Por convención, el lado servidor suele esperar a que el cliente inicie la conversación.

La opción *proxyarp* indica al servidor, caso de estar conectado a otras redes, que debe coger paquetes que vayan destinados al nodo cliente, realizando funciones de pasarela. En nuestro caso, "gárgola" debe saber que los paquetes que viajan por la red 1.0.0.0 con destino a "fénix" deben ser recogidos por el servidor, para ser reencaminados por la línea serie.

Por último, la opción *"gárgola:fénix"* indica que el nodo de este lado se llama 'gárgola', y que el interlocutor es 'fénix'.

EL CLIENTE PPP

La configuración en el cliente es muy similar. Utilizaremos DIP para conectar y lanzar el interlocutor. Siguiendo el ejemplo del servidor PPP de los párrafos anteriores, el cliente debería invocar el siguiente texto para DIP:

```
# Ejemplo de script DIP: ppp.dip

# Inicializar nombres de nodo local y remoto
  get $local fénix
  get $remote gárgola

# Conectaremos a una red A (1.0.0.0)
  netmask 255.0.0.0

# Inicializar línea /dev/modem y velocidad
# 19200
  port modem
  speed 19200

# Marcar número
  send 1234567\r
# Si no aparece "login:" en 45 segundos, hay un
# error
  wait login: 45
  if $errlvl!=0 goto error
# en caso afirmativo, introducir login
  send ppp\r

# hemos conectado, lanzar modo PPP
  print Conexión realizada...
  default
  mode PPP
  exit

error:
# Hubo algún error, abortar
  print Problema en la conexión
  quit
```

Como vemos, la conexión es prácticamente idéntica al caso de SLIP. La opción *default* indica que todos los paquetes IP que se generen en el cliente se encaminarán por la línea serie hacia el servidor. Como el cliente es un nodo suelto, todas las peticiones de conexión irán, por lógica, hacia el nodo que hace de pasarela para él. Repase el comentario que realizamos al respecto al hablar del comando 'route' si no le queda claro el por qué de esta opción.

A pesar de que este fichero es muy similar al que se ha visto con SLIP, ahora tendremos un problema adicional, como es disponer de un fichero */etc/ppp/options* en el nodo local que sea compatible con la configuración del servidor. En nuestro ejemplo:

```
asyncmap 200A0000
crtscts
mru 296
netmask 255.0.0.0
-pap
-chap
modem
fénix:gárgola
```

Puede observar que han desaparecido algunas opciones con respecto a la configuración del servidor. Las opciones *silent* y *passive* debe estar en uno y sólo uno de los lados, como se ha comentado anteriormente. La opción *proxyarp* solo tiene sentido en el lado que puede actuar como pasarela, y normalmente el cliente será un nodo aislado, de forma que se elimina.

Por otra parte, el orden de nombrar ambos lados (*fénix:gárgola*) es el inverso al caso anterior. Se nombra primero el nodo local, y luego el remoto.

En el caso de una conexión PPP, la compatibilización de opciones en el cliente y el servidor es algo más complicada que en un enlace SLIP, y requiere alguna prueba adicional antes de tener todo configurado de forma óptima. Para nuestra ayuda, tendremos dos opciones que podemos incluir en el fichero */etc/ppp/options*, como son:

```
debug
kdebug 2
```

Con estas opciones, PPP vuelca información sobre el proceso de conexión, y posibles causas del fallo de ésta, por medio del demonio *syslog*. (normalmente el resultado de este volcado se encuentra en */var/adm/messages* y */var/adm/syslog*). Si tiene problemas en la conexión, y cree que el problema ocurre después de que DIP invoque a

PPP, estas opciones le serán de inestimable ayuda para saber en qué punto dejan de estar de acuerdo ambos interlocutores. El parámetro que acompaña a la opción *kdebug* indica el nivel de detalle de los volcados ('1' indica mínimo detalle).

Para concluir una conexión PPP, de forma parecida al caso anterior, se invoca el siguiente comando:

dip -k

COMPROBACIÓN DE RESULTADOS

Tanto al realizar conexiones con SLIP como con PPP, el programa DIP se encarga de "ocultar lo feo". Debe observar que se realizan para usted las siguientes operaciones:

- Diálogo con el módem,
- Establecimiento de conexión,
- Programación de la línea en modo binario (IP)
- Asociación de la línea al interfaz (ppp0 ó sl0) adecuado; y
- Actualización de la tabla de rutas

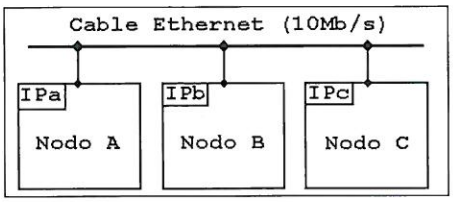


Figura 1: La red Ethernet.

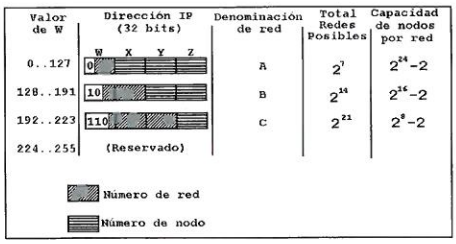


Figura 2: Las direcciones IP.

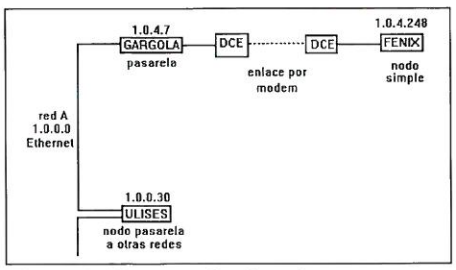


Figura 3: El escenario ejemplo.

Dado que, entre otras cosas, el proceso de configuración con *ifconfig* y *route* se realiza de forma automática, puede comprobar los conceptos vistos con los siguientes comandos:

ifconfig ppp0

En caso de tener establecida una conexión PPP, al teclear este comando podemos observar el estado de la conexión.

Para conexiones SLIP, el comando es equivalente:

ifconfig sl0

Observará algo similar que con el comando anterior antes de que la conexión se efectúe. Compruebe las estadísticas (RX y TX) en las dos últimas líneas. Conforme invoque programas que empleen TCP/IP, estas estadísticas variarán. La primera columna indica los paquetes que se han cursado con éxito. El resto de números contabilizan errores por diferentes causas. Si no obtiene comunicaciones correctas, y se incrementa el número de errores, probablemente hay un conflicto en las tablas de rutas. Compruebe este punto con el comando:

route

Como puede adivinar, si va a usar SLIP/PPP para conectar a un nodo servidor del que no conoce su configuración (principalmente, de un proveedor de acceso a Internet), necesitará que el responsable de este sistema le indique las opciones correctas, y le asigne una dirección IP y *netmask,* antes de poder realizar conexiones de forma satisfactoria. Como consuelo, el esfuerzo de configurar con éxito todo este galimatías se verá premiado con un mundo de posibilidades, como son la comunicación por correo electrónico, la lectura de tablones (*news*) de cualquier tema que le interese, y "bajarse" ficheros de cualquier lugar del mundo sin mayor problema que elegir el más interesante. En definitiva intercambiar conocimientos y experiencia con quien se le antoje.

CONCLUSIONES

De todo lo comentado en este artículo, debemos extraer que, aunque la típica conexión en modo terminal es la más sencilla, nos permite aprovechar las características de la línea serie muy pobremente.

Por el contrario, proveedores de conexión a Internet (como Goya, Servicom, IBM, algunas BBS y muchos otros que aparecerán en su ciudad en previsiblemente poco tiempo) permiten la conexión bien mediante SLIP o PPP, convirtiendo a su ordenador en un nodo más, con plenos derechos de envío y recepción de información mediante paquetes IP. Y conforme aumente la competencia entre proveedores, las tarifas de conexión empezarán a ser asequibles.

Aunque no se ha tratado en profundidad la configuración de un sistema como cliente de cualquiera de estos protocolos, esperamos que el lector haya aprendido conceptos básicos que le serán fundamentales para entender lo que se ofrece en estos servicios. De igual modo, debe saber que no es necesario un software especial para realizar este tipo de conexiones. Puesto que ambos protocolos son estándar, usted sólo necesitará que le asignen su propia dirección IP y averiguar el modo de automatizar la conexión e invocar al interlocutor SLIP/PPP del otro lado de la línea.

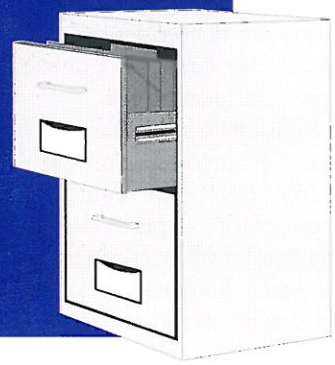
Por otro lado, algunos lectores pueden intentar configurar un ordenador en el trabajo con el que poder conectar desde casa como un nodo más dentro de la red corporativa.

O, simplemente, puede conectar de forma más robusta ordenadores remotos mediante un módem, sin necesidad de esperar a "bajarse" un fichero para poder teclear otros comandos, o leer el correo, o todo lo que usted imagine.

En definitiva, y hasta que la RDSI nos alcance, es el aprovechamiento más completo que se puede realizar con su módem a un nivel superior. Esperamos que lo disfrute en toda su extensión.

CONCEPTOS BÁSICOS (2ª PARTE): BASES DE DATOS

Juan Manuel y Luis Martín



Como ya se mencionó en el artículo del mes pasado, Clipper es un lenguaje de programación orientado al manejo y la gestión de bases de datos. Por tanto, dispone de gran número de herramientas creadas con este fin. Las bases de datos manejadas por Clipper son de tipo relacional, y su formato es uno de los más extendidos dentro del mundo de la microinformática. Se trata del formato de bases de datos .DBF, compatible con el de dBASE, salvo en los ficheros de índice. Sin embargo, Clipper también incorpora librerías que permiten el manejo de bases de datos en otros formatos (Paradox 3.5, FoxPro 2, dBASE III PLUS y dBASE IV). Es importante destacar también que Clipper permite la utilización simultánea de varias bases de datos y el establecimiento de vínculos entre ellas.

Clipper trabaja con bases de datos relacionales que sigan el estándar XBase

DEFINICIÓN Y ESTRUCTURA

Las bases de datos pueden definirse como estructuras de datos donde se almacena información relativa a entidades del mundo real. Esta información se almacena en forma de propiedades o atributos que caracterizan cada una de las entidades. En el caso de Clipper, la información se almacena en un fichero con extensión .DBF, cuya estructura es similar a una tabla. Cada fila contiene la información relativa a una entidad, mientras que cada columna contiene las propiedades de la misma. Las filas o entidades se denominan *registros* y las columnas o propiedades se denominan *campos*. Los registros se identifican con un número, mientras que los campos lo hacen por su nombre.

El número de registros de una base de datos es variable, mientras que el número de campos es fijo, y serán los mismos para todos los registros. Los campos pueden ser de cualquiera de los cinco tipos de datos que maneja Clipper (Caracteres, Numérico, Fecha, Lógico y Memo).

El diseño de una base de datos consiste en la especificación de su estructura, es decir, en la definición de cada uno de sus campos. Para ello deben especificarse cuatro parámetros por cada campo:

- Nombre
- Tipo
- Tamaño
- Decimales

El nombre identifica el campo y consta de 10 caracteres, sin espacios en blanco, el primero de los cuales debe ser una letra. El tipo especifica la clase de datos que va a contener ese campo. El tamaño indica la longitud del campo en caracteres. Los campos de Fecha y Lógico tienen tamaño fijo (8 y 1, respectivamente), mientras que los campos Memo tienen una longitud variable.

Como continuación del artículo del mes pasado, el presente artículo también pretende ofrecer una introducción a los conceptos básicos de la programación en Clipper; en este caso se analizan los conceptos relativos a las bases de datos.

Por último, si el campo es Numérico, debe especificarse el número de decimales que contendrá.

CREACIÓN

Cuando los programas se limitan a manejar bases de datos ya existentes, será necesario utilizar una herramienta externa que permita la creación de las mismas. Con Clipper se suministra la aplicación *DBU*, que permite crear, visualizar y modificar bases de datos.

Para la creación de una base de datos desde dentro de un programa se utiliza una *matriz estructural*, en la que se almacena la estructura que tendrá la base de datos previamente a su creación. Se trata de una matriz de submatrices, cada una de ellas con cuatro elementos:

- Una cadena con el nombre del campo.
- Un carácter con el tipo del campo.
- Un número con la longitud del campo.
- Un número con los decimales del campo.

La tabla 1 muestra los caracteres correspondientes a cada tipo de datos. Por su parte, para los campos Memo debe especificarse longitud 0. Una vez creada la matriz estructural, debe llamarse a la función *DBCREATE()*, especificando en sus argumentos el nombre del fichero de la base de datos y la matriz estructural.

```
DBCREATE( cBaseDatos, aEstructura )
```

APERTURA Y CIERRE

Antes de poder trabajar con una base de datos, es necesario abrir el fichero. Igualmente, después de trabajar, será necesario cerrar el fichero. La apertura se realiza mediante el siguiente mandato:

```
USE [xcBaseDatos][READONLY]
```

En *xcBaseDatos* se indica el nombre del fichero de base de datos (literalmente o como expresión de tipo carácter encerrado entre paréntesis). La cláusula *READONLY* permite abrir el fichero sólo para lectura. Si no se indica ningún argumento se cierra la base de datos que se encuentre abierta.

MOVIMIENTO

Cada uno de los registros se identifica por un número que coincide con la posición que ocupa en la base de datos (y con el orden en que fue introducido). Como cualquier otro fichero, cuando se abre una base de datos se crea un *puntero*. En este caso, el puntero de una base de datos contiene el número del registro actual. De este modo, el acceso a los datos se realiza a través del registro actual. Para acceder a otro registro será necesario alterar el puntero, es decir, cambiar el registro actual.

Por defecto las bases de datos tienen la extensión .DBF y los índices .NTX

Para alterar el puntero, se utilizan mandatos y funciones de movimiento, que pueden clasificarse en dos tipos:

- De posicionamiento: Permiten situar el puntero en un determinado registro atendiendo a su posición.
- De búsqueda: Permiten situar el puntero en un determinado registro atendiendo al contenido de sus campos.

La tabla 2 muestra los mandatos y funciones de posicionamiento, y la tabla 3 muestra los de búsqueda.

MANIPULACIÓN

Las operaciones de manipulación de los datos pueden resumirse en las siguientes:

- Introducción de nuevos datos.
- Modificación de datos existentes.
- Borrado de datos existentes.

Para añadir un registro nuevo a una base de datos, primeramente se añade un registro en blanco (con sus campos vacíos) al final de la base de datos.

TABLA 1

Letra	Tipo de datos
C	Carácter
N	Numérico
D	Fecha
L	Lógico
M	Campo Memo

Para ello se utiliza el siguiente mandato:

```
APPEND BLANK o DBAPPEND()
```

Posteriormente, se rellenan los campos con los nuevos valores, utilizando sentencias de asignación o mandatos especiales para ello.

Para alterar el contenido de los campos de un registro, será necesario posicionarse sobre él. Posteriormente, los valores pueden modificarse mediante el uso de sentencias de asignación (:=).

Clipper proporciona un mandato que permite modificar los valores de los campos de varios registros simultáneamente:

```
REPLACE [ámbito]
  idCampo1 WITH expr1
  [, idCampo2 WITH expr2...]
  [WHILE lCondición1]
  [FOR lCondición2]
```

Los registros de una base de datos no se borran directamente. Primero se realiza un *marcado* de los registros que se desea borrar (*), utilizando para ello el siguiente mandato:

```
DELETE [ámbito] [FOR lCondición1]
  [WHILE lCondición2]
```

De está forma, es posible *desmarcarlos*, haciendo el proceso reversible. Para ello se utiliza el siguiente mandato:

```
RECALL [ámbito] [FOR lCondición1]
  [WHILE lCondición2]
```

Para eliminar físicamente los registros marcados debe realizarse un *empaquetado* de la base de datos. Este proceso se realiza con el mandato *PACK*. Existe otro mandato, *ZAP*, que permite borrar todos los registros, estén marca-

TABLA 2	
Función	Descripción
RECNO()	Devuelve el número del registro actual
LASTREC()	Devuelve el número total de registros
DBGOTO(n)	Posiciona el puntero en el registro n
DBGOTOP()	Posiciona el puntero en el primer registro
DBGOBOTTOM()	Posiciona el puntero en el último registro
DBSKIP(n)	Salta n registros a partir del actual
EOF()	Indica si el puntero ha sobrepasado el final del fichero
BOF()	Indica si el puntero ha sobrepasado el principio del fichero

Funciones y mandatos de posicionamiento.

dos o no. Estas dos operaciones deben realizarse con precaución, ya que sus resultados son irreversibles.

FILTROS

Clipper proporciona un método para ocultar los registros que no cumplen una determinada condición, de forma que los mandatos y funciones de movimiento ignoren dichos registros. Para ello, debe especificarse una condición lógica, denominada *filtro*. La especificación de un filtro se realiza con el siguiente mandato:

SET FILTER TO [lCondición]

Si no se especifica ningún argumento, la condición de filtro que esté vigente quedará desactivada.

LOS ÍNDICES

En la mayoría de las ocasiones, los usuarios de una base de datos desean acceder a los registros según un criterio o un orden determinado que, por lo general, no coincidirá con el orden físico que poseen los registros. La solución a este problema la proporcionan los *índices*.

Los índices ordenan la información contenida en una base de datos en función de un criterio determinado. Dicho criterio se basa en la aplicación de una expresión a cada uno de los registros de la base de datos, ordenándose éstos en función del valor devuelto. La expresión de ordenación se conoce como *clave de indexación*.

La ordenación de los registros no se realiza de forma física, es decir, los registros de la base de datos seguirán almacenados en el mismo orden. La ordenación se realiza en un fichero auxiliar, denominado *Fichero de índice* o simplemente *Índice*. Su contenido serán los distintos valores de la clave de indexación ordenados y el número del registro de la base de datos que debe ocupar ese lugar. Además, una misma base de datos puede tener varios índices, cada uno de ellos con criterios distintos. De esta forma, es posible disponer de la información de la base de datos ordenada según diferentes criterios. La creación de un fichero de índice se realiza mediante el siguiente mandato:

INDEX ON expClave TO xcÍndice
 [UNIQUE]

El nombre del fichero de índice se indica en *xcÍndice*, literalmente o como expresión de caracteres encerrada entre paréntesis. Debe tenerse en cuenta que para crear un índice de una base de datos, ésta debe encontrarse abierta.

APERTURA Y CIERRE

Al igual que con las bases de datos, los archivos de índice deben abrirse antes de utilizarse y cerrarse cuando se termine de trabajar con ellos. La apertura de un archivo de índice no tiene porque realizarse en el mismo momento que la base de datos asociada, aunque sí es recomendable. Esto se debe a que todas las variaciones que se realicen en una base de datos únicamente se reflejarán en aquellos archivos de índice que se encuentren abiertos. De este modo, si se elimina un registro de la base de datos, todos aquellos índices que no se encontrasen abiertos en ese momento no reflejarán el cambio producido.

La apertura de un índice puede realizarse por tanto de dos formas. La primera consiste en añadir una cláusula al mandato USE:

USE [xcBaseDatos]
 INDEX xcListaÍndices

De esta forma, es posible indicar todos los archivos de índice que se desea abrir con la base de datos (hasta un total de 15), separándolos por comas.

La otra forma consiste en abrir los índices asociados a una base de datos que ya se encuentra abierta. Para ello debe utilizarse el mandato siguiente:

SET INDEX TO [xcListaÍndices]

El cierre de una base de datos supone el cierre de todos los índices asociados a la misma. Si se desea cerrar los índices pero no la base de datos, puede utilizarse el mandato *SET INDEX ON* sin argumentos.

SELECCIÓN DE ÍNDICES

Aunque una base de datos tenga abiertos varios índices a la vez, sólo uno de ellos será el *índice activo*. De esta forma, a pesar de que todos los cambios de la base de datos se reflejen de automáticamente en todos los ficheros de índice abiertos, la base de datos sólo puede estar ordenada según un criterio a la vez.

Cuando se abren los índices, se crea automáticamente una lista en la que cada uno de ellos se identifica por el número de orden con el que fue abier-

Los índices permiten ordenar las bases de datos según distintos criterios

TABLA 3	
LOCATE	Posiciona el puntero en el primer registro que cumpla cierta condición
FOUND()	Indica si una búsqueda con LOCATE ha tenido éxito
CONTINUE	Pasa al siguiente registro que cumpla la condición del LOCATE

Funciones y mandatos de búsqueda.

to. Inicialmente, el primer índice de la lista será el índice activo. Para cambiar de índice activo puede utilizarse el mandato siguiente:
SET ORDER TO [nÍndice]

BÚSQUEDA Y POSICIONAMIENTO

La gran ventaja que ofrecen los índices no sólo consiste en la posibilidad de disponer de toda la información de la base de datos ordenada, sino que además las búsquedas de datos pueden realizarse de una forma mucho más rápida. Para ello, debe utilizarse un mandato de búsqueda distinto, basado en la clave de indexación:

SEEK expresión

Este mandato busca y posiciona el puntero en el primer registro cuya clave de indexación coincida con el valor devuelto por *expresión*. Para saber si la operación ha tenido éxito debe consultarse el valor devuelto con la función *FOUND()*.

RECONSTRUCCIÓN DE ÍNDICES

Como ya se ha explicado anteriormente, cuando se producen cambios en una base de datos y alguno de los índices asociados no se encontraba abierto, éste no habrá actualizado la información que contiene, por lo que dará problemas y errores.

La solución a este problema podría consistir en volver a crear todos los índices asociados a la base de datos. Sin embargo, este proceso es muy lento. Clipper proporciona un mandato especial, *REINDEX*, que permite reconstruir los archivos de índice. Este mandato reconstruye todos los ficheros de índice que se encuentran actualmente abiertos.

Mediante los filtros se posibilita el ocultar registros que no cumplan cierta condición

LAS ÁREAS DE TRABAJO

Cuando se trabaja con bases de datos relacionales, a menudo es habitual la necesidad de trabajar con varias bases de datos simultáneamente. Clipper proporciona un método para realizar este tipo de operaciones. Para ello, cuando se arranca un programa, la memoria se divide en zonas denominadas *áreas de trabajo*. Cada una de estas áreas de trabajo está preparada para contener una base de datos. De esta forma, es posible tener abierta una base de datos en cada una de las áreas de trabajo.

Las áreas de trabajo se identifican mediante un número que varía entre 1 y 255, aunque las cinco últimas son internas de Clipper. Un área de trabajo

EL FUTURO ESTÁ EN SUS MANOS

Este libro le transportará, en un viaje sin precedentes, a través del sistema operativo que revolucionará todos los ordenadores del mundo

WINDOWS 95
NO SE QUEDE ATRÁS
PÍDALO AL TELF.: (91) 741 26 62

TABLA 4

Función	Descripción
SELECT(Alias)	Devuelve el número del área de trabajo identificada por *Alias*
ALIAS(n)	Devuelve el alias del área de trabajo número n
DBSELECTAREA()	Realiza la misma operación que el mandato SELECT
DBCLOSEAREA()	Cierra la base de datos abierta en el área de trabajo actual

Funciones de manipulación de las áreas de trabajo.

se encontrará libre u ocupada en función de si hay o no una base de datos abierta en ella.

Cuando se abre una base de datos puede añadirse una serie de cláusulas al mandato USE con el fin de colocar la base de datos en distintas áreas de trabajo.

```
USE [xcBaseDatos]
  [INDEX xcListaÍndices]
  [ALIAS xcAlias] [NEW] [READONLY]
```

Si se incluye la cláusula *NEW*, la base de datos se abrirá en la siguiente área de trabajo que esté libre. Si se incluye la cláusula *ALIAS*, es posible asignar un nombre al área de trabajo, de forma que no será necesario conocer su número.

En un momento dado sólo puede haber un área de trabajo activa. De esta forma, todas las operaciones realizadas irán referidas a la base de datos que se encuentre abierta en el área de trabajo activa. Para cambiar el área de trabajo activa se utiliza el siguiente mandato:

```
SELECT xnÁrea | xcAlias
```

El argumento puede, por tanto, ser un número de área de trabajo o bien un alias. La tabla 4 incluye una relación de las funciones que proporciona Clipper para manejar las distintas áreas de trabajo.

EL OPERADOR ALIAS

Clipper posibilita la realización de operaciones en un área de trabajo distinta de la actual sin necesidad de cambiar de área. Esta posibilidad sólo se encuentra disponible para expresiones (no para mandatos). Para ello, se utiliza un operador especial denominado operador *alias*.

El operador alias se representa mediante un guión seguido de un signo de mayor que (->). En la parte izquierda del operador puede indicarse un área de trabajo (bien por el alias de la misma o por su número). En la parte derecha se indica la expresión a aplicar a dicho área de trabajo, encerrada entre paréntesis. De esta forma, la expresión será ejecutada como si el área activa fuese la que se encuentra delante del operador.

```
xcAlias->(expresión)
xnÁrea->(expresión)
```

Este operador puede utilizarse también para cualificar una variable. De esta forma, es posible indicar delante del nombre de la variable si ésta se trata de una variable de memoria o por el contrario se trata de un campo de una base de datos.

```
FIELD->idCampo
MEMVAR->idVariable
```

RELACIONES

Como se ha venido mencionando a lo largo del artículo, Clipper maneja bases de datos relacionales. Esto quiere decir que existe la posibilidad de relacionar varias bases de datos, permitiendo la construcción de estructuras más complejas.

Las bases de datos se enlazan unas con otras mediante lo que se conoce como *relaciones*. En una relación, una de las bases de datos será la base de datos *padre* y la otra será la base de datos *hija*. Para establecer relaciones entre dos bases de datos, lo primero que debe obtenerse es una expresión aplicable a los registros de la base de datos padre. Dicha expresión se denomina *expresión de relación*. La peculiaridad de esta expresión es que los valores devueltos por la misma deben ser compatibles con la clave de indexación de la base de datos hija.

De esta forma, un movimiento dentro de la base de datos padre provoca de manera automática que se produzca una operación similar a un *SEEK* del valor devuelto por la expresión de relación en la base de datos hija.

Para establecer relaciones entre las bases de datos de distintas áreas de trabajo debe utilizarse el mandato:

```
SET RELATION TO
  [expRelación1 INTO xcAlias1]
  [TO expRelación2 INTO xcAlias2...]
  [ADDITIVE]
```

De esta forma se establecerán relaciones entre la base de datos del área de trabajo actual (o base de datos padre) y las bases de datos de otras áreas de trabajo (bases de datos hija). En los argumentos *expRelaciónX* se indica la expresión de relación encargada de unir la base de datos padre con la base de datos del área de trabajo identificada por *xcAliasX*.

Una vez establecida una relación, todas las operaciones de movimiento que se realicen en la base de datos padre se verán reflejadas en la (o las) base de datos hija.

Si no se especifican argumentos en el mandato SET RELATION TO, se cancelan todas las relaciones existentes.

Las relaciones permiten crear estructuras compuestas por varias Bases de Datos

BIBLIOGRAFÍA

"CA-Clipper 5.2"; L. y J.M. Martín Caballero; Publicaciones R.P.H.

"CA-Clipper: Guía del programador"; Computer Associates

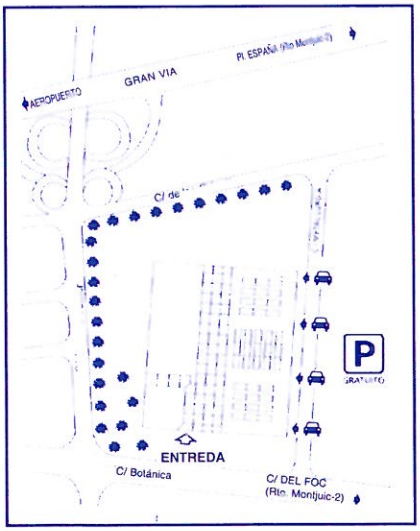
TOWER

ESTAMOS EN

Informat 95

"THE GLOBAL VILLAGES"

DEL 13 AL 16 DE SEPTIEMBRE (ambos inclusive)



EN MONTJUIC-2, EL NUEVO RECINTO FERIAL DE BARCELONA

SORTEOS DIARIOS EN NUESTRO STAND D-1

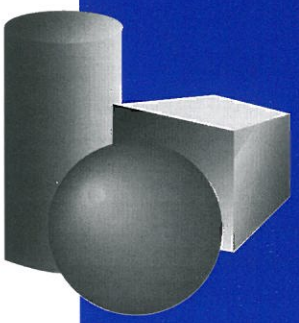
BBS's POR PROVINCIAS

SISTEMA	LOCALIDAD	OPERADOR	TELÉFONO	BPS	F t U	LMXJVSD
ALBACETE						
DataTouch BBS	Albacete	Demolition	967-228336	2.400	. O .	(00-08)
Psycodelic BBS	Albacete	Sigmund Freud	967-506962	14.400	. O 1	(23-08)
ALICANTE						
Akra Leuka BBS	Alicante	Eloy Cortés	96-5227365	14.400	. O .	(00-24)
Cyberiada	Alicante	Carlos	96-5201279	28.800	. O .	(00-24)
Europa III BBS	Valencia	Francisco	96-3460109	19.200	F Z 2	(00-24)
Loom BBS	Valencia	Alfonso	96-2512782	16.800	. . .	(00-24)
No Man's Land HST	Elche	Orlando	96-6661325	21.600	. . .	(00-24)
Policeman BBS	Alcira	Anubis	96-2402976	14.400	S H .	(00-24)
Zas-Two BBS	Elche	Juande	96-5425968	14.400	. . .	(00-24)
ALMERÍA						
Telebase	Almería	Francisco	950-274830	21.600	F H .	(00-24)
ASTURIAS						
Asturcom BBS	Gijón	Luis	98-5162118	14.400	. . .	(00-24)
Asturias BBS	Oviedo	Ricardo	98-5272773	14.400	. . .	(00-24)
Cero BBS	Oviedo	Armando	98-5222635	28.800	F O 1	(00-24)
Techno City #1	Oviedo	Javier	98-5272284	28.800	. O 2	(00-24)
AVILA						
Happycomm BBS	Avila	Javier	920-254433	28.800	F H .	(00-24)
BALEARES						
Europa IV BBS	Palma	Jaime	971-740397	14.400	. O 2	(00-24)
Ibosim BBS	Ibiza	Serafín & Fco.	971-346020	19.200	F Z .	(00-24)
Nura BBS	Mahón	Juan Ruiz	971-353689	14.400	F H .	(00-24)
Todocom BBS	Ibiza	Daniel	971-311757	16.800	F Z .	(00-24)
BARCELONA						
Abaforum.(10 lin)	Sant Cugat	David	93-5893888	14.400	. . .	(00-24)
Activity BBS	Granollers	Pedro	93-8464202	14.400	. . .	(00-24)
Amiga Hypnosis	Barcelona	Joan Sola	93-2968708	14.400	. . .	(00-24)
Amiga Penedes BBS	Vilafranca	Ferrán	93-8923983	14.400	. . .	(00-24)
Axel's BBS	Barcelona	Axel	93-2036846	14.400	. . 1	(09-13)
BBS Moon S.T.	Hospitalet	J. Antonio	93-3376200	28.800	. . .	(00-24)
BlackAdders' BBS	Barcelona	Jordi	93-2844630	14.400	. . .	(00-07)
Bucolico BBS	Casteldefels.	Javier	93-6362177	14.400	. . .	(00-24)
Cinema BBS	Barcelona	Jesús	93-4177367	16.800	. . .	(00-24)
Gotham City	Hospitalet	Víctor	93-4492929	16.800	F Z .	(09-23)
Hallowed BBS	Barcelona	Mario	93-6628729	14.400	. . .	(18-06)
Iluro	Barcelona	Josep	93-7994013	14.400	. . .	(00-24)
Lone Star OS/2	Mataró	Fidel	93-7906630	14.400	. . .	(00-24)
Micaco BBS I	Badalona	David Casas	93-3846169	14.400	. O .	(00-24)
Murphy BBS	Ripollet	Jordi J	93-5808515	28.800	. H 2	(00-24)
Natural World BBS	Barcelona	Alex	93-2780556	14.400	. . .	(00-24)
Nexus Msx BBS	Barcelona	Ángel	93-4252171	14.400	. . .	(00-24)
Spiral	Viladecans	Toni	93-6370949	14.400	. . .	(00-24)
BIZKAIA						
+Bytes BBS	Tudela	Julio	94-8410490	14.400	. . .	(00-24)
Deusto BBS	Bilbao	Javier	94-4130397	16.800	. Z .	(00-24)
Kender(L1)	Bilbao	Oskar	94-4763506	14.400	. . .	(00-24)
Serpent BBS	Algorta	Oscar	94-4605666	16.800	F . .	(00-24)
CÁDIZ						
People's BBS	Cádiz	Chico Medina	956-283031	14.400	. O .	(00-08)
CANTABRIA						
Cantabria-1	Santander	Manuel	942-281222	21.600	. . .	(00-24)
CASTELLÓN						
Costa Este BBS	Benicarló	Ángel	964-461204	14.400	. . .	(00-24)
CEUTA						
Hércules BBS	Conil Front	Jose María	956-443226	14.400	. O .	(00-24)
Hobby's BBS	Ceuta	Braulio	956-516859	14.400	F . .	(00-24)
CIUDAD REAL						
Alcazar BBs	Alcazar S.J	José	926-550903	28.800	F O 1	(00-24)
STigma V	Ciudad Real	Francisco	926-222614	28.800	F O .	(00-24)
CÓRDOBA						
El Zoco BBS	Córdoba	Santiago	957-439292	14.400	F O 2	(00-24)
LA CORUÑA						
Edi BBS	Santiago	José	981-502110	19.200	F Z 1	(00-24)
Zombi BBS	La Coruña	Vicente	981-226558	14.400	. O .	(00-24)
GIPUZCOA						
Buenavista BBS	S. Sebastián	Pablo	943-352220	14.400	. O .	(00-24)
Junior BBS	S. Sebastián	Alfredo	943-310370	14.400	. O .	(00-24)
Sanse BBS I	S. Sebastián	Paco	943-471688	28.800	F . .	(00-24)
GERONA						
Floppy BBS	Arbucies	Jordi Cornet	972-162528	14.400	O .	(20-02)
GRANADA						
Atlantis I	Granada	Alfredo	958-123848	14.400	. . .	(00-24)
BlasterSound BBS	Granada	Miguel	958-293583	19.200	F Z .	(00-24)
La Abadía BBS	Granada	J. M. Guglieri	958-204855	16.800	F Z .	(00-24)
HUELVA						
Colón-93	Huelva	Frank	959-319287	28.800	. S 2	(00-24)
Onuba-1	Huelva	Manolo	959-553199	21.600	. . .	(00-24)
JAÉN						
Dragón BBS	Jaén	Alfonso	953-272584	19.200	F Z .	(00-24)
LA RIOJA						
EIC BBS	Logroño	Rafael	941-262720	14.400	. . .	21-08TT
LAS PALMAS						
Bikom-BBS	G. Canaria	Jech Gines	928-381095	21.600	. . .	(00-24)
The Home of Thunder.	Las Palmas	Octavio	928-648057	21.600	F H .	(00-24)
LEÓN						
Chrome BBS	León	Antonio	987-242585	14.400	. . .	(20-02)
LUGO						
Dactonio BBS	Monforte	Antonio	982-400792	9.600	. . .	(00-24)
Galicia-1	Monforte	Manuel	982-410902	21.600	F H 2	(00-24)
LLEIDA						
Agron BBS	Lleida	Sisco	973-239918	14.400	. . .	(00-24)
Savage Club BBS	Lleida	Jordi	973-220440	14.400	. . .	(00-24)
MADRID						
Abc analog #1	Madrid	José Delgado	91-6345042	21.600	. H .	(00-24)
Alhambra NetWorks	Madrid	Enrique	91-3149442	14.400	. O .	(00-24)
Amiga_Support	Madrid	Eugenio	91-5187229	14.400	. . .	(00-24)
Atenea BBS	Madrid	Jorge	91-5042898	14.400	. O .	(00-24)
Atharva	Madrid	Manuel	91-3770996	14.400	. . .	(00-24)
Athos BBS	Madrid	Julián	91-6323014	14.400	F O 1	(00-24)
Camelot BBS	Madrid	Jesús A.	91-5012652	14.400	. O .	(18-24)
Cash-Line BBS	Madrid	Alfredo	91-5191336	14.400	. . .	(00-24)
Corben	Madrid	Enrique	91-5702555	14.400	. . .	(00-24)
CRC OK	Madrid	Los Usuarios	91-4151215	14.400	. . .	(00-24)
Deckard BBS	Madrid	Pedro	91-6431067	14.400	. . .	(18-06)
Digital Dreams Multimedia	Madrid	Emilio	91-5197575	14.400	. . .	(00-24)
Dulces SueñOS/2	Madrid	Guillermo	91-5199835	28.800	. S 2	(00-24)
Edison BBS	Madrid	Sancho	91-6340224	14.400	. O .	(00-24)
Enchufe-OS/2	Madrid	Santiago	91-5477210	28.800	. . .	(00-24)
Espíritu Navideño	Madrid	José Eugenio	91-6930276	14.400	. O .	(00-24)
Europa BBS OS/2	Madrid	Francisco	91-3208460	19.200	F Z 2	(00-24)
GuruMeditation	Madrid	Javier	91-3831317	28.800	. O .	(00-24)
Hiades BBS	Madrid	Angel	91-6387660	14.400	. . .	(24-08)
Hispania BBS	Madrid	Sergio	91-6636889	28.800	F . .	(00-24)
Hispania II BBS	S. S. Reyes	Efraim	91-6636337	14.400	F H .	(00-24)
Icaro	Madrid	Emilio	91-5194645	14.400	. . .	(00-24)
Kroandal BBS	Madrid	Carlos	91-5484566	14.400	. . .	(23-10)
La Calavera Feliz	Madrid	Ruben	91-8138161	28.800	F O .	(00-24)
La Escuela BBS	Madrid	Victor	91-5014702	14.400	F O 2	20-09TT
La Radio BBS	Madrid	Fernando	91-3461196	28.800	. O .	(00-24)
Lineas Aéreas BBS	Madrid	Carlos	91-6580879	28.800	. . 1	(00-24)
Magma BBS	Madrid	Alan	91-3615543	14.400	. H 2	(00-24)
Mail-Simons	Madrid	Benito	91-5195238	14.400	. . .	(00-24)
MAS. (Amiga)	Madrid	Eugenio	91-5187229	28.800	. O .	(00-24)
Mañana Más	Madrid	Rafa G.	91-6680368	28.800	. . .	(01-10)
Mercurio BBS	Madrid	Vicente	91-5258090	19.200	. . .	(00-24)
Nada de Nada	Madrid	Guillermo	91-6882919	9.600	. . .	(00-24)
Najarra	Alcalá	Alberto	91-8815795	14.400	. H 1	(00-24)
Nebula BBS	Madrid	Paco	91-5495210	14.400	. . .	(00-24)
NidOS BBS	Madrid	Manuel	91-5094237	14.400	. . 1	(00-24)
Nimey BBS	Madrid	Rafael	91-5480364	14.400	. O .	(00-24)
Omega.I BBS	Madrid	Oliver	91-6375531	14.400	. . .	(00-24)
Periodismo Elect	Madrid	Fernando	91-4115724	14.400	. . .	(00-24)
Programer's Land	Madrid	Francisco	91-3065142	14.400	. . .	(00-24)
Psico BBs	Madrid	Javier	91-7419171	14.400	. . .	(23-02)
Rafa STD-1	Madrid	Rafael	91-5447282	14.400	. . 1	(00-24)
Sakery Fox	Madrid	Alberto	91-4139855	14.400	. . .	(00-24)
Santuario BBS	Madrid	Pedro	91-5608060	14.400	. . .	(00-24)
Stigma I	Madrid	Patrick	91-8523510	28.800	. . 1	(00-24)
Tercera Generacion	Madrid	Casimiro	91-5227184	14.400	. . .	(00-24)
Tex-Mex BBS	Madrid	Santiago	91-5193232	14.400	. . .	(00-10)
TimeBBs	Madrid	Miguel	91-3671141	14.400	. . .	(00-24)
Venus BBS	Madrid	Fermín	91-6838259	28.800	F O 1	(00-24)
Virtual R. (Amiga)	Madrid	Vicente	91-5523292	28.800	. O .	(00-24)
MÁLAGA						
bAuHaUs	Málaga	Luis	95-2213374	19.200	F Z .	(00-24)
New Order BBS	Torremolinos	Carlos	952-381343	19.200	F Z .	(00-24)
MELILLA						
Anduin	Melilla	Bartolomé	95-2674815	14.400	F . .	(00-24)
MURCIA						
A&A	Murcia	Juan	968-306473	14.400	. O .	(00-24)
Central BBS	Murcia	Felman	968-342640	28.800	. O .	(00-24)
Club PC Murcia BBS	Murcia	Riky	968-201819	14.400	. . 2	(00-24)
Club PC Murcia II	Murcia	Riky	968-201262	14.400	. . 2	(00-24)
Europa VII BBS	Murcia	Andrés	968-241628	14.400	. O 2	(00-24)
Ghost in the Night	Yecla	Juan Muñoz	968-751605	14.400	F H .	(20-07)
NAVARRA						
Konty BBS	Pamplona	David	948-312790	14.400	. . 2	(22-08)
Pergola BBS	Pamplona	Ricardo	948-185344	28.800	F S 1	(00-24)
OURENSE						
Revolta BBS	Carballino	Manuel	988-274777	21.600	F H .	(00-24)
PONTEVEDRA						
La Aldea Global	Sanxenxo	Juan	986-691525	19.200	F Z .	(00-24)
Radio Maceda BBS	Vigo	Tony Saa	986-234969	14.400	. O .	(00-24)
Rias Baixas BBS	Pontevedra	Carlos	986-863434	28.800	F O .	(00-24)
SALAMANCA						
Cisem BBS	Salamanca	Francisco	923-121554	28.800	F O .	(00-24)
Colossus BBS	Salamanca	Andrés	923-210974	14.400	. . .	(00-24)
STA. CRUZ DE TENERIFE						
Active Sense	Tenerife	Jose María	922-344359	14.400	. . .	(00-24)
Einstein	Tenerife	Luis Aguirre	922-200353	14.400	. . .	(00-24)
Halloween BBS	Tenerife	Eduardo	922-203155	14.400	. . .	(00-24)
Symelis BBS	Tenerife	Juan Carlos	922-751392	14.400	. . .	(00-24)
SEVILLA						
Agujeros Negros	Sevilla	Joaquín	95-4333864	28.800	F O .	(00-24)
Alas	Sevilla	Angel	95-4530358	19.200	F Z .	(00-24)
Alija	Sevilla	Luis	95-4160113	16.800	F . .	(00-24)
El Vicio del Silicio	Sevilla	Isidro	95-5666106	19.200	F Z .	(00-24)
Elektra	Sevilla	Federico	95-4161749	16.800	F Z .	(00-24)
Glastombury BBS	Sevilla	Julián	95-4277973	19.200	F Z .	(00-24)
Infovideo BBS	Sevilla	Arturo	95-4537273	14.400	. . .	(00-24)
La Voix Al Harem	Sevilla	Alfonso	95-4275321	16.800	F Z .	(00-24)
La Voix Al Maharif	Sevilla	Alfonso	95-4275081	14.400	. . .	(00-24)
Viernes 13	Sevilla	Manolo	95-4539132	19.200	F Z .	(00-24)

NOTAS: BSP = Max. Vel. t = (F=FAX, Z=Zyxel, H=HST) U=mes última confirmación
HORAS: T=00-24, .=No activo, A=14-24, B=20-23

LA GESTIÓN DE EVENTOS

Ignacio Cea



Lo que caracteriza a un sistema de ventanas no es que en cada una de ellas se puedan visualizar diferentes cosas sino, más bien, que en cada una de ellas se pueden ejecutar distintas operaciones con resultados dispares. Así una ventana puede estar esperando la pulsación de un botón OK para cerrarse mientras que los muñecos situados en la del fondo parecen moverse ajenos a esa situación.

Cualquiera que observe puede llegar a tener la absoluta certeza que el sistema operativo de la máquina sobre lo que observa esto es multitarea, pues aparentemente es capaz de realizar diversas tareas de forma simultánea. Ni que decir tiene que el MSDOS no es multitarea y que cualquier efecto de este tipo en una aplicación construida sobre él es, completamente, simulado.

UN SISTEMA MULTITAREA

Olvídese, por un momento, de las limitaciones de su MSDOS y piense en un sistema operativo multitarea cualquiera como bien pueden ser OS2 o UNIX en los que varios procesos pueden ejecutarse paralelamente.

Este esquema es útil y válido para aprender los conceptos básicos de la gestión de eventos en un sistema de ventanas de una manera más rápida y sencilla. Tras esto lo migraremos a un sistema monotarea como MSDOS.

EL SERVIDOR DE VENTANAS

Cabe fácilmente imaginar en este contexto, que las diversas ventanas del sistema no son sino diferentes procesos. Lo que se hace, por tanto, en una no tiene nada que ver con lo que se realiza en otra y ambas cosas, además, pueden ser hechas de forma simultánea.

Las distintas órdenes del usuario (eventos) tales como pulsación de teclas, movimiento de ratón, etc, pueden ser recogidas por otro proceso cuyas misiones fueran: buscar aquella ventana adecuada para su interpretación, mandárselo y ponerse, nuevamente a la escucha. El nuevo proceso actúa, pues, como un servidor no sólo de sus órdenes sino también de las ventanas que son, por otro lado, los consumidores últimos de las mismas. (Figura 1).

LAS VENTANAS

Dentro de ese esquema, la estructura principal del código gestor de la ventana ha de ser, básicamente, un bucle. Ese bucle ha de, o bien recoger los eventos mandados por el servidor y gestionarlos (en caso de que estos estuvieran disponibles), o bien dedicarse a cualquier otra tarea mientras tanto.

Las ventanas analizan el evento mandado por el servidor dentro del MainLoop

Como símil, imagine que usted ha puesto a cargar las baterías de su cámara de vídeo mientras escribe una carta con su procesador de textos. Su ocupación fundamental es la carta; sin embargo, periódicamente, se levantará a mirar si la carga ha finalizado (evento) para, en ese caso, desconectar el cargador y poner la pilas en su máquina. Tanto si hay evento como si no tras mirar y, quizás, gestionar, proseguirá con su carta. (Figura 2). Ese bucle tan importante

Un sistema de ventanas no sería prácticamente nada si el usuario no pudiera realizar distintas acciones sobre ellas. Este capítulo aborda el complicado mundo de la gestión de eventos y en él aprenderá los principios básicos para emular las capacidades de los más poderosos sistemas multitarea.

dentro del mundo de las interfaces gráficas suele conocerse bajo el nombre de *MainLoop*.

UNA VENTANA PREDILECTA

En este momento surge la cuestión sobre la forma en la que el servidor de ventanas busca aquella adecuada para interpretar la orden recibida del usuario (se ha pulsado una tecla). Sin duda, que los algoritmos son muchos, y aquí no hay ni tiempo ni espacio para reparar en todos ellos. No queda otra solución que tratar uno de ellos.

Si se fija un poco notará que la mayor parte de los sistemas de ventanas que actualmente existen en el mercado, marcan el contorno de una de ellas de una forma distinta al resto. Mientras usted no indique lo contrario será ésta y no otra quien reciba las órdenes del usuario y trate de interpretarlas. En Windows, por ejemplo, dicha ventana está marcada con un color distinto en la parte superior de la misma. A esa ventana la llamaremos ventana principal. (Figura 3).

LA GESTIÓN DE EVENTOS

Cuando el evento llega a la ventana pueden suceder dos cosas: o bien que el evento es reconocido e interpretado por la ventana, o bien es catalogado como desconocido. En el primer caso el servidor se pondría nuevamente a la espera de nuevos eventos; en el segundo, por el contrario, tendría que ir mandándolo al resto de las ventanas hasta que, o bien alguna de ellas lo reconociera o bien se agotaran, y el evento fuera definitivamente descartado.

El orden en el que ese evento es mandado al resto de ventanas es el que aquellas tienen dentro de una pila de ellas (Figura 4) que el sistema mantiene en todo momento (ver capítulos anteriores).

LAS VENTANAS TAMBIÉN HABLAN

En un sistema de ventanas complejo, las ventanas no se limitan a ser meros intérpretes de órdenes, sino que también han de ser capaces de darlas.

Imagine que dentro de una ventana existe un botón con la leyenda "Cerrar" y que el usuario presiona el botón izquierdo del ratón estando situado sobre él. El servidor recibiría el evento y se lo mandaría a la ventana. Aquella lo recibiría y procedería a realizar la acción de cierre. Sin embargo, un objeto no puede borrarse a si mismo, por lo que no tendría más remedio que pedir al servidor de ventanas que lo hiciera por ella. (Figura 5).

Las ventanas no sólo emplean al servidor para pedirle su destrucción sino para muchas otras cosas, como mandar eventos a otras ventanas o pedir que sean ascendidas dentro de la pila del sistema, hasta convertirse en privilegiadas (lo que vulgarmente se llama poner una ventana delante de todas).

SERVIDOR C++

A lo largo de estos tres últimos artículos se ha descrito la clase "Ventana" como alguien que tenía un conocimiento sobre todas aquellas ventanas que integraban el sistema en un momento dado.

Es precisamente esta característica la que la convierte, también, en la candidata idónea para realizar las tareas del servidor de ventanas descrito anteriormente. La función sería desempeñada por el método estático "servidor" declarado dentro de ella. (Figura 6).

FIGURA 6

```
class Ventana {
        private :
                static NodoVentana *Inferior;
                static NodoVentana *Superior;
                static void zonas_ocultas (void);
                static void zonas_vistas (void);
                void dibujar_antiguas_zonas_ocultas (void);

                ZonasEscritura ZonasTapadas;
                ZonasEscritura ZonasVistas;
                int X1,Y1,X2,Y2;

        public :
                Ventana (int x1,int y1,int x2,int y2);
                ~Ventana ();
                void mover (int ax,int ay);
                void reescalar (int ax,int ay);
                virtual void dibujar (objeto *);
                virtual int gestor_eventos (void) = 0;

                static Evento CSEvento;
                static void servidor (void);
};

class Evento {
        private :
                int Codigo;
                Ventana *Propietaria;
                int Parametro1;
                int Parametro2;

        public :
                Evento (int Id,Ventana *win,int P1,int P2);
                ~Evento () { /* Vacio */ }
                int codigo (void) const;
                Ventana *propietaria (void) const;
                int parametro1 (void) const;
                int parametro2 (void) const;
};
```

Las clases.

El interprete de la ventana es un método virtual puro

INTERPRETES C++

La función de intérprete puede realizarse mediante un método no estático dentro de cada ventana al que se llamará "gestor_eventos". Dado que la función de dicho gestor dependerá, según lo visto, del tipo de ventana que reciba en evento el método habrá de ser declarado como virtual (Figura 6).

FIGURA 8

```
Servidor de ventanas
void Ventana::servidor (void)
{
        Ventana que gestiona = primera ventana
        Mientras no se quiera salir
                Si hay que releer el evento
                        CSEvento = LeerEvento
                Resultado = Gestionar evento por primera ventana
                Releer evento en la próxima pasada
                En caso de Resultado
                        Recibido e interpretado
                                Empezar el bucle
                        a Gestionar por el Servidor
                                Si el evento es finalizar el sistema
                                        Salir = VERDAD
                                Si el evento es destruir
                                        borrar CSEvento.Propietaria
                                        Ventana que gestiona = primera ventana
                                Si el evento es ascender
                                        ascender CSEvento.Propietaria
                                        Ventana que gestiona = primera ventana
                                Si el evento es mandado a otra
                                        Gestionar evento por CSEvento.Propietaria;
                        Evento desconocido
                                Si ha sido enviado a todas las ventanas del sistema
                                        Ventana que gestiona = primera ventana;
                                si no
                                        Ventana que gestionar = siguiente en la pila
                                        No releer evento
                Fin de en caso de...
        Fin del mientras..
}

Gestión de eventos en las ventanas
int Ventana ::gestor_eventos (void)
{
        En case de CSEvento.Codigo
                Movimiento del ratón
                ...
                        Devolver evento recibido e interpretado
                Pulsación de una tecla (ejemplo)
                        Si la tecla es la de salir
                                CSEvento.Codigo = finalizar el sistema
                                Devolver a Gestionar por el Servidor
                        Si no
                                Devolver evento recibido e interpretado
                ...
        Fin de en caso de...

        Devolver Evento desconocido
}
```

El pseudocódigo.

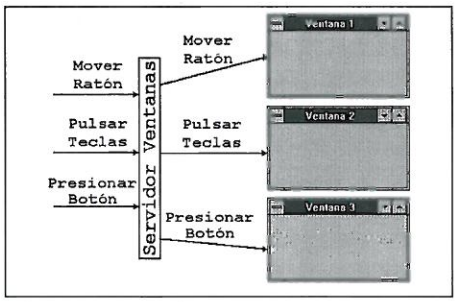


Figura 1: El servidor de ventanas.

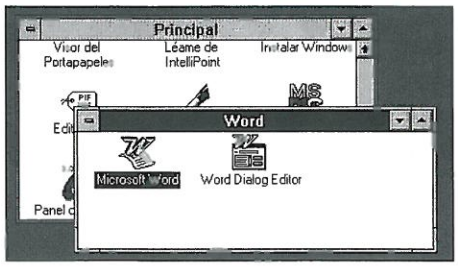


Figura 3: La ventana principal.

Para construir una ventana de usuario no hay más que derivar de la clase "Ventana" la nuestra, y rellenar el cuerpo del método "gestor_eventos" según lo que se desea que haga (Figura 7).

INFORMACIÓN

La información que fluye entre el servidor (método "servidor") y el intérprete de eventos de cada ventana (método "gestor_eventos") es el evento.

El evento puede ser caracterizado por un código, un puntero a la ventana que ha de ser el destinatario del mismo y un par de variables más que hicieran las veces de parámetro.

En un primer vistazo, podría pensarse que el puntero a la ventana sobra, ya que el servidor conoce perfectamente la ventana a la que se envía un evento; sin embargo es muy importante a la hora de considerar la posibilidad de que sean las ventanas las que manden eventos a otras, en cuyo caso no hay más remedio que poder especificar un destinatario para que el servidor tenga la posibilidad de redirigirlo adecuadamente.

FIGURA 7

```
class MiVentana : public Ventana {
        private
                ...
        public :
                ...
        virtual int gestor_eventos (void); /* ya no
es pura */
};
```

Una ventana de usuario.

Hay muchos eventos que van acompañados de parámetros. Así, por ejemplo, el evento de "tecla pulsada" ha de ir acompañado del código de la tecla en cuestión. Los dos parámetros adicionales sólo tienen como fin albergar esos datos adicionales.

UNICIDAD DEL EVENTO

La información que fluye entre las distintas ventanas y el servidor de ellas, cada vez que éste recibe un evento del usuario es común a todas ellas. Es decir, que la información es la misma hasta que, o bien es tratada por alguna ventana, o bien rechazada por el servidor. Esto obliga a pensar en un objeto estático de la clase ventana (común a todos los objetos) para representar los eventos, tal y como muestra la figura 6.

RESPUESTAS DEL GESTOR DE EVENTOS

El gestor de eventos pensado puede dar tres tipos de respuestas: el evento ha sido reconocido e interpretado, el evento es desconocido para esta ventana y el evento ha sido alterado, y debe ser nuevamente tratado por el servidor de ventanas. Cada una de esas respuestas puede ser codificada con un número, razón por la cual el método gestor_eventos devuelve un entero. Entre la última de las respuestas podemos contemplar tanto la posibilidad de que la ventana pida su destrucción como la de que esa misma esté dirigiendo un mensaje a otra.

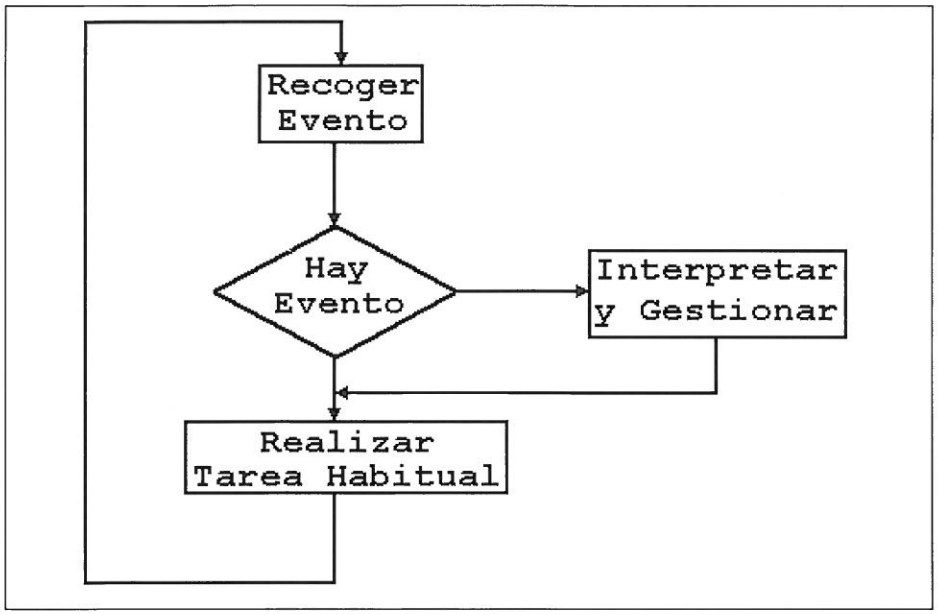


Figura 2: El mainLoop.

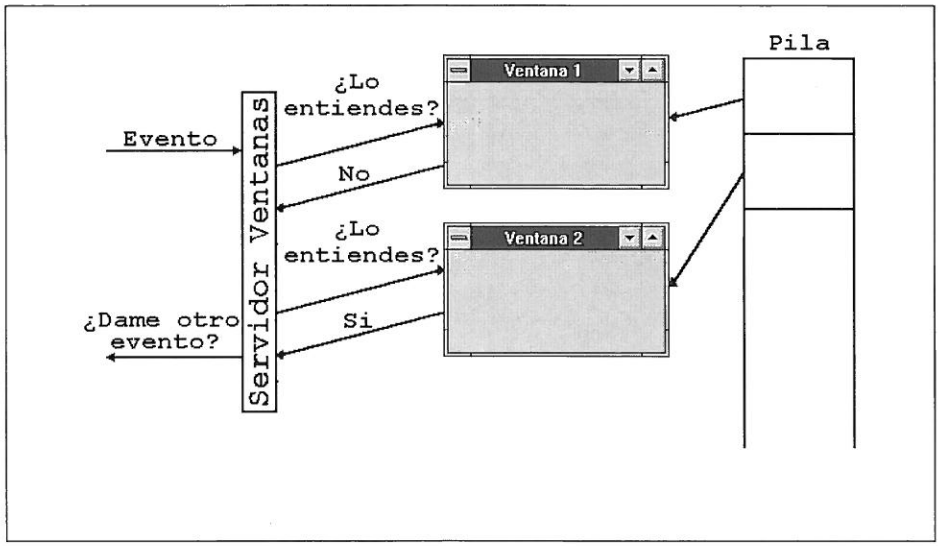


Figura 4: La gestión de eventos.

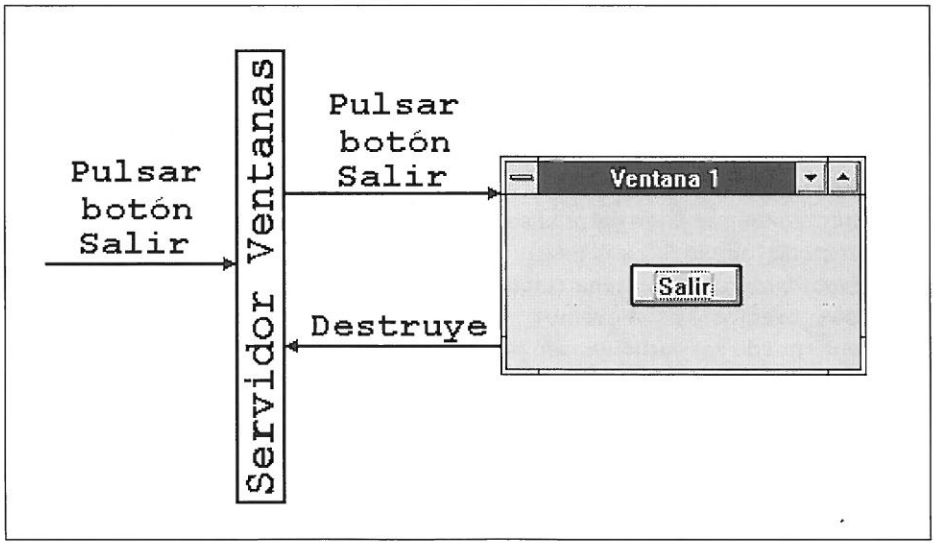


Figura 5: Las peticiones de las ventanas.

EL NO EVENTO

De entre todos los eventos cabe la posibilidad de considerar uno que indicará que no ha ocurrido nada, y que pueda ser empleado por la ventana que lo reciba para realizar algún tipo de operación cíclica (como mover un fondo) mientras espera alguno más interesante.

UNA LIMITACIÓN

Si se fija un poco podrá ver un enorme fallo dentro del planteamiento hasta ahora seguido. Si cualquiera de los gestores de las distintas ventanas del sistema se entretuviera en exceso con el tratamiento del evento mandado por el servidor, todo el sistema se bloquearía ya que, por un lado el servidor no recuperaría el control y, por el otro las otras ventanas no recibirían nada que gestionar.

Esta situación no se da, evidentemente, dentro de un sistema multitarea en el que el sistema operativo concedería *slots* de tiempo a cada uno de los procesos, hicieran aquellos lo que hicieran. Como mucho se conseguiría que un montón de eventos se apilaran dentro de la ventana para su gestión.

Por tanto, si quiere simular de verdad un sistema multitarea, hemos de evitar estas situaciones haciendo que todos sus procesos de gestión de eventos sean de ciclo abierto.

Tampoco tiene sentido que consideremos la posibilidad de que los eventos puedan ser apilados, ya que un evento no puede ser tratado hasta que no lo sea el anterior. La única manera viable de solucionar este problema dentro de un sistema operativo monotarea sería, desarrollar el servidor mediante el uso de interrupciones algo que, evidentemente, escapa al alcance de este curso didáctico.

EL CÓDIGO

Sólo resta exponer el pseudocódigo que rige el funcionamiento de los dos métodos principales: "servidor" y "gestor_eventos". Ambas cosas pueden observarse en la figura 8.

PRÓXIMA ENTREGA

Las bases están sentadas sólo queda unirlas y realizar un pequeño ejemplo con el que clarificar todas las ideas expuestas.

CONTENIDO DEL CD-ROM

VELOCIS 1.3.0
WINDOWS STANDALONE

Junto al presente número de la revista se incluye un CD-ROM con la versión 1.3.0 Windows Standalone de Velocis. Se trata de un gestor de bases de datos Cliente/Servidor diseñado para crear aplicaciones de bases de datos en los lenguajes C y C++. La versión ofrecida por Sólo Programadores viene, totalmente gratis, con una licencia de uso para un desarrollador. Los términos de la licencia se pueden encontrar en el propio directorio de Velocis.

Además, en el CD-ROM se suministra la distribución Slackware 2.2.0.1 del popular sistema operativo Linux. Esta versión incluye el kernel 1.2.3 de Linux, las librerías libc 4.6.27 y la versión 3.1.1 del entorno gráfico X-Window Xfree86.

Por otro lado, como viene siendo habitual, se incluye una selección de programas shareware y los ficheros que acompañan a los artículos de la revista.

VELOCIS

En el directorio \VELOCIS se encuentra preinstalado el sistema gestor de bases de datos de Raima Corporation. Para ejecutar Velocis correctamente se recomienda copiarlo al disco duro tal y como está almacenado en el CD-ROM. Esto se puede hacer mediante el comando:

```
xcopy <Unidad CD-ROM>\velocis <Unidad destino>\velocis /s /e
```

Donde <Unidad CD-ROM> es la unidad de disco correspondiente al CD-ROM (por ejemplo, d:) y <Unidad destino> es el disco duro en el que se desea instalar Velocis (ejemplo: c:), que deberá tener al menos 5 megas de espacio libre. El comando xcopy preguntará si <Unidad destino>\velocis es un fichero o un directorio, a lo que se responderá con lo segundo. Una vez realizada la copia es preciso asegurarse de que se ha creado el directorio SYSLOG dentro del directorio de Velocis. La estructura de directorios del sistema es la siguiente:

- BIN: Todas las librerías de enlace dinámico (DLL) y los ejecutables.

- INCLUDE: Ficheros de cabecera para la programación en C y C++.

- LIB: Librerías de Velocis.

- CATALOG: Base de datos del catálogo del sistema.

- SYSLOG: Directorio del dispositivo de registro.

- SQLDB: Directorio del dispositivo de base de datos SQL de ejemplo.

- EXAMPLES: Ejemplos de Velocis.

- EXAMPLES\SQL: Ejemplos de módulo de extensión, funciones SQL definidas por el usuario (UDF) y procedimientos SQL definidos por el usuario (UDP).

- EXAMPLES\RSQL: Fuentes de la utilidad RSQL y ejemplos de definición de bases de datos.

- EXAMPLES\RDM: Ejemplos de manejo de base de datos a bajo nivel.

- INTRO: Documentos con información referente a Velocis y Raima.

Para finalizar, hay que crear un pequeño fichero de configuración en el directorio de Windows mediante cualquier editor de textos ASCII. El nombre del fichero es rds.ini y su función es la de indicar a Velocis, entre otras cosas, la ubicación de su catálogo. Debe contener las siguientes líneas:

```
[RDS]
CATPATH=<Unidad>\velocis\catalog
```

Donde <Unidad> será la unidad donde Velocis ha sido copiado (en el ejemplo anterior, c:).

Para realizar sus funciones, Velocis utiliza una serie de ejecutables situados en el directorio BIN. Los más importantes para el usuario son los siguientes:

- admin.exe: Utilidad de administración de Velocis.

- rsql.exe: Versión QuickWin de la utilidad interactiva de SQL.

- dals.exe: Versión QuickWin de la utilidad de acceso a datos (DAL).

- sddlp.exe: Versión QuickWin del procesador del lenguaje de manejo de datos SQL (SQL DDL).

- ddlprocs.exe: Versión QuickWin del procesador del lenguaje de manejo de datos Velocis.

- _dsrv.exe: Driver MS-DOS del dispositivo virtual (VXD).

- rsql.exe: Versión VXD de la utilidad RSQL.

- dal.exe: Versión VXD de la utilidad DAL.

Para comprobar que la instalación es correcta, el lector puede realizar los siguientes pasos:

1.- Ejecutar la utilidad de administración admin (admin.exe en el directorio BIN de la instalación de Velocis).
2.- Conectarse al servidor. Para ello, abrir el menú Server y seleccionar la opción Login. Acto seguido, teclear el nombre del servidor (en la versión Standalone cualquier nombre es válido), admin como identificador de usuario y secret como palabra de paso. El usuario admin está creado previamente por Velocis y tiene privilegios de administrador. Otros usuarios son Wayne y guest, con privilegios de usuario normal, cuyas palabras de paso son Father y guest respectivamente.
3.- Una vez se ha logrado la conexión, la ventana del servidor mostrará una lista con las bases de datos disponibles. Entre ellas aparecerán syscat, sales, invntory y tims.
4.- Seleccionar la opción Accounts del menú View. Ahora, en lugar de la lista de bases de datos, se verá una lista con los usuarios del servidor. Deben aparecer al menos tres: Wayne, admin y guest.
5.- Por último, seleccionar la opción Devices del menú View. Entre otros, aparecerán los dispositivos catdev, samples y sysdev.

Si no hubo problemas, Velocis puede acceder al catálogo del sistema. Para salir del servidor, se puede utilizar la opción Exit del menú Server.

Para más información, consultar los ficheros en formato de ayuda de Windows de Velocis situados en el directorio de Velocis. Son los siguientes:

- ***admref.hlp:*** Este fichero proporciona la información necesaria para instalar, configurar y aprender las principales tareas de administración del sistema Velocis.

- ***license.hlp:*** Contiene los términos de la licencia de uso de la versión 1.3.0 Windows Standalone de Velocis.

- ***rdsapi.hlp:*** Fichero de referencia sobre las APIs, el lenguaje de definición de datos (DDL) y el lenguaje de acceso a datos (DAL).

- ***readme.hlp:*** Información general sobre Velocis.

- ***speced.hlp:*** De forma análoga al fichero admref.hlp, este fichero expone la instalación y las pautas básicas de manejo del sistema.

- ***sqlref.hlp:*** Se trata de una guia del lenguaje SQL con información sobre las APIs para la utilización de SQL desde C.

En el directorio INTRO se puede encontrar información adicional. Los archivos están en formato Word para Windows. A continuación se describe brevemente el contenido de los mismos:

- ***cmrcdint.doc:*** Es un documento de introducción al sistema gestor de bases de datos.

- ***ramcm1.doc:*** Proporciona información sobre los creadores de Velocis.

- ***rom3bro.doc:*** En este documento se describe otro de los productos de Raima Corporation.

- ***velcomp.doc:*** Comparación entre las características de Velocis y otros sistemas gestores de bases de datos.

- ***velocis1.doc:*** Es un extenso manual que abarca las diversas características de este producto.

LINUX

En el directorio LINUX del CD-ROM se encuentra la distribución Slackware de Linux. La instalación de este sistema operativo ya ha sido expuesta con detalle en números anteriores de la revista. El lector puede consultar al respecto los números 4, 6 y 10, y el libro "Linux, Unix para todos" que acompañaba al número 9 de Sólo Programadores, además de los documentos incluidos en el directorio de Linux.

SHARE

Los programas shareware del CD-ROM están almacenados en distintos subdirectorios que cuelgan del directorio SHARE. Entre ellos se encuentra un editor de sprites, utilidades para Clipper y utilidades para Windows. Entre ellas cabe destacar:

-Top Security es una utilidad orientada a la protección de datos.

-Seguro 3.0 es una librería para la protección de aplicaciones realizadas en Clipper 5.x , con la que se puede evitar que las utilidades sean pirateadas con programas como Rescue, Unclip, Valkirie etc.

-Scroll permite ver como un juego o programa que funciona a 320 x 200 modifica la pantalla realizando un scroll de la misma.

-Cgrup es una aplicación para crear grupos de programas con sus iconos en el administrador de programas.

-Cobolbas son dos utilidades para los desarrolladores en el lenguaje COBOL.

REVISTA

El directorio REVISTA contiene los directorios CTHREADS, CURSOP, DEMOS, LINUX, SONIDO, VTX y WINPROG. Cada uno de ellos contiene los archivos que acompañan a los artículos de la revista.

CORREO DEL LECTOR

En esta sección, los lectores de SÓLO PROGRAMADORES tienen la oportunidad de hallar respuesta a los problemas que puedan tener en cualquier tema relaccionado con la programación.

Hola, en primer lugar, felicidades por la serie de artículos sobre formatos gráficos, pues somos muchos los programadores más interesados en los gráficos y los juegos, que en la gestión. Sólo encuentro una carencia importante, el formato FLI o FLC. Creo que está muy extendido y sería interesante que explicaran cuál es su formato y uso.

David Ramos González
(Málaga)

Tiene razón, son (somos) muchos los programadores más dedicados al aspecto gráfico o musical de los ordenadores. El formato FLC no sólo es un formato gráfico, como bien sabes se trata de un formato que almacena *frames* o partes de una pantalla gráfica, las cuales al irse dibujando continuamente, crean una animación. Realmente se podría utilizar exclusivamente como un formato gráfico, pues bastaría con guardar sólo una imagen de gráfico.

Su formato es bastante complicado y no se puede explicar en esta sección, pero tomamos nota para un artículo a corto plazo.

Tengo un CD-ROM MITSUMI IDE ATPI y el kernel de Linux no lo soporta, pero en las utilidades incluidas en el CD-ROM que incluyeron en su revista, venían los fuentes de los drivers para mi CD. ¿Cómo puedo compilarlos?

Rafael Romero
(Sevilla)

Depende de cada caso, pero seguro que junto con los ficheros fuente se incluye un fichero con la información para su generación mediante el comando *make*.

Me interesó mucho el artículo que publicaron sobre programación de la tarjeta Sound Blaster. En la actualidad necesito acceder a ella, pero sin utilizar ningún driver para DOS o Windows, es decir, programando directamente el DSP. Necesito información completa sobre los comandos del mismo, y no consigo encontrar el paquete de desarrollo para Sound Blaster (Developer Kit for Sound Blaster Series). Podrían publicar algo al respecto. Gracias de antemano.

Antonio J. Tomás
(Murcia)

Sobre la programación de la Sound Blaster o compatible "a pelo", estamos (y seguiremos) publicando artículos referentes al tema. La figura 1 contiene una lista de los comandos del DSP utilizado en las tarjetas Sound Blaster.

He leído el artículo sobre la programación de la memoria extendida (XMS). Aparentemente, mi ordenador sólo dispone de memoria expandida (EMS). ¿Cómo puedo acceder a ella? ¿Que diferencias hay entre los dos tipos?

Sean Hartnoll
(Barcelona)

Es raro que su ordenador sólo disponga de memoria expandida, debe tratarse de un 286. Habitualmente toda la memoria se toma como extendida y luego se configura parte o toda como expandida. Incluso hoy en día es normal el tener toda la memoria como extendida y como expandida simultáneamente, dejando a las aplicaciones la elección del tipo. Este proceso suele hacerse de la siguiente manera:

1.- Arranque del ordenador
   - Memoria "limpia" o *raw*
2.- HIMEM.SYS
   - Memoria XMS
3.- EMM386, QEMM o similar
   - Memoria EMS y memoria XMS

Acabo de instalar Linux y tengo algunas dudas por resolver:

1.- Al arrancar el sistema no lo hace con la configuración de teclado en español, así que siempre tengo que configurarlo manualmente mediante el programa setup. ¿Cómo puedo activarlo permanentemente?
2.- ¿Cómo puedo saber el espacio libre que me queda en la partición Linux?
3.- ¿Cómo puedo acceder a una disquetera o a un CD-ROM desde Linux?

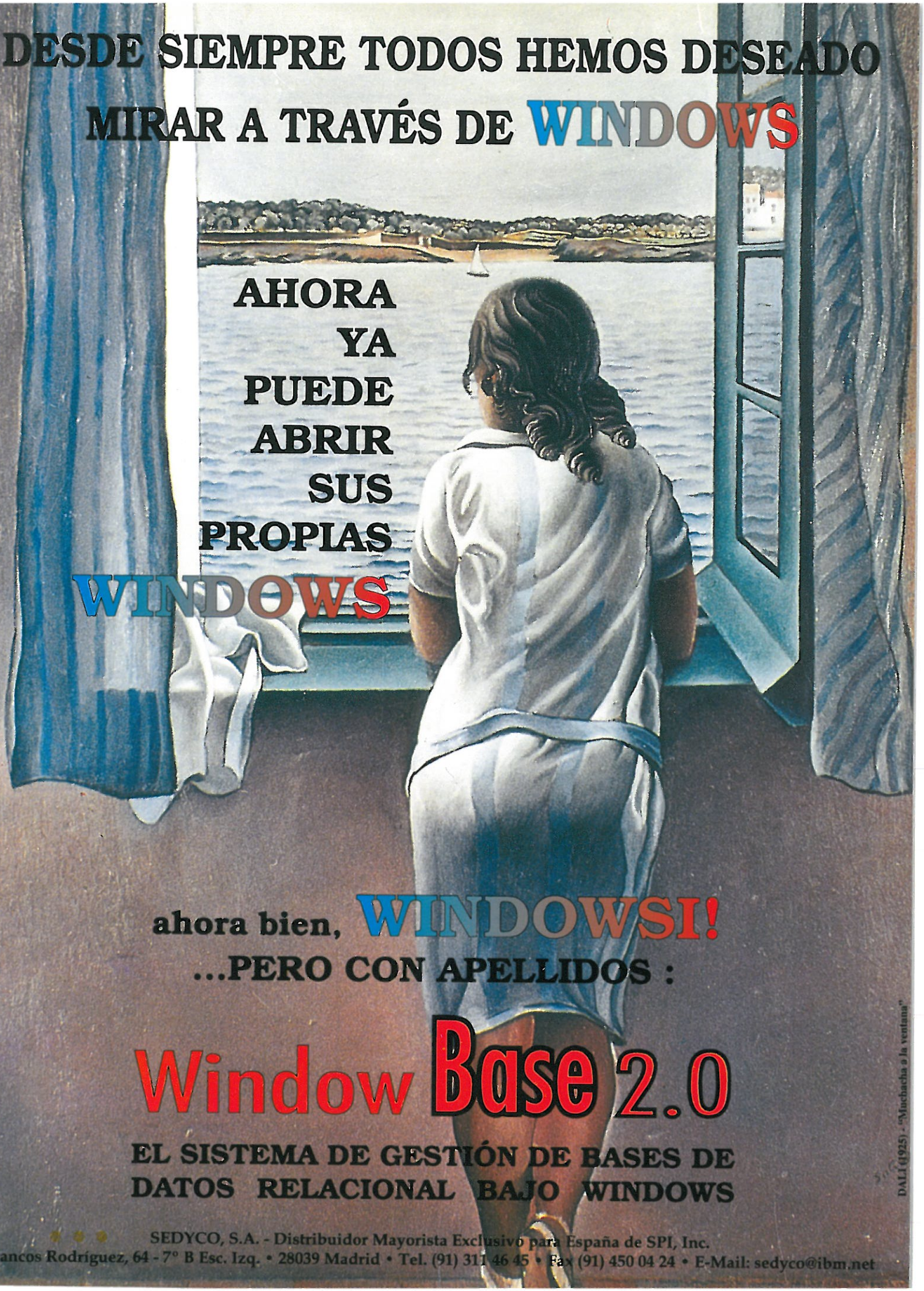
Adolfo Alandro García
(Madrid)

Vayamos por pasos: - La configuración o *mapeo* del teclado se encuentra en el fichero */etc/rc.d/rc.keymap* y en dicho fichero debe encontrarse la línea */usr/bin/loadkeys /usr/lib/kbd/keytables/es.map*.

- Para conocer la ocupación y el espacio libre de la partición, teclee *df* y le mostrará toda la información de los distintos sistemas de archivos, en bloques de 1 Kb.

- Para poder acceder a una disquetera o a un CD-ROM, lo primero es *montarlos* en el sistema. Por ejemplo para *montar*

DESDE SIEMPRE TODOS HEMOS DESEADO MIRAR A TRAVÉS DE WINDOWS



ahora bien, WINDOWSI!

...PERO CON APELLIDOS :

Window Base 2.0

EL SISTEMA DE GESTIÓN DE BASES DE DATOS RELACIONAL BAJO WINDOWS

DALÍ (1925) - "Muchacha a la ventana"

© SEDYCO, S.A.

SEDYCO, S.A. - Distribuidor Mayorista Exclusivo para España de SPI, Inc.
Francos Rodríguez, 64 - 7° B Esc. Izq. • 28039 Madrid • Tel. (91) 311 46 45 • Fax (91) 450 04 24 • E-Mail: sedyco@ibm.net